195 PTAS.
(IVA Incluido)

mi computer

113



P.V.P. Canarias, Ceuta y Melilla 185 Ptas.

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen X-Fascículo 113

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 120 fascículos de aparición semanal, encuadernables en diez volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-183-6 (tomo 10)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 188603
Impreso en España-Printed in Spain- Marzo 1986

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (120 fascículos más las tapas, guardas y transferibles para la confección de los 10 volúmenes) son las siguientes:

a) Un pago único anticipado de 27 105 ptas. o bien 10 pagos trimestrales anticipados y consecutivos de 2 711 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.





Una educación integral

Tony Sleep

La introducción de los ordenadores en los planes de estudio escolares en los años setenta y ochenta ha sido tema de intenso debate tanto entre los maestros como entre los padres. Aquí analizaremos el cambiante contenido de los cursos de informática durante la pasada década y demostraremos que las tendencias actuales indican que tales cursos tienen un futuro limitado.

Al introducirse la informática en los planes de enseñanza de los países europeos (a finales de los años sesenta) no se consideró necesaria la presencia de los ordenadores en las aulas. Por este motivo no fue sorprendente que los primeros exámenes de la nueva asignatura tuvieran un marcado acento en lo teórico, dejando en segundo término las aplicaciones prácticas. Por supuesto, los alumnos de entonces no tenían acceso a los potentes micros y periféricos de hoy en día, que les hubieran permitido crear sofisticados sistemas "de la vida real".

Durante los años setenta, la informática adquirió notable importancia, llegando a situarse junto a las asignaturas más tradicionales.

Los planes de estudio relativos a informática varían de un país a otro, pero los maestros más "inspirados" desde el punto de vista educativo suelen incluir materias tales como "crear una conciencia del impacto de los ordenadores..." y "desarrollar aptitudes para las comunicaciones y la resolución de problemas..." En los años ochenta, se le restó trascendencia a la enseñanza de programación (especialmente BASIC) como parte fundamental del curso, pasando a utilizar paquetes existentes para resolver problemas. Irónicamente, se podría argumentar que un conocimiento de un lenguaje de programación quizá constituya el aspecto más vocacional de tal curso, si bien obviamente esto dependería de la elección del lenguaje. Además, los aspectos de "informática" de la asignatura (puertas lógicas, sumadores, etc.) desaparecieron casi por completo y la dirección de los cursos en su conjunto se orientó hacia el usuario y se apartó de los aspectos técnicos del hardware.

Conciencia vocacional
La popularidad del estudio de la informática como tal no está creciendo tan rápidamente como pudiera pensarse. Cada vez más los alumnos están abandonando su estudio a medida que alcanzan la edad de dejar la escuela, prefiriendo combinar un conocimiento elemental de la informática con otros conocimientos, tales como contabilidad, diseño y administración

En Gran Bretaña, en 1985, se intentó unificar los diversos planes de estudio existentes (véase recuadro de p. 2242). Una parte esencial del curso implicaba la resolución de problemas: el proyecto práctico que se requiere que complete cada candidato. Como cabía esperar, el formato de este proyecto se ha alterado considerablemente con el correr de los años, tendiendo a una situación en la que se solicita al alumno que plantee y resuelva un problema utilizando un ordenador. Esta solución puede suponer la utilización de paquetes existentes, la escritura de programas nuevos o una combinación de ambos. Un proyecto simple podría ser el de crear un sistema sencillo para conservar los archivos de videos y prestatarios de una biblioteca ambulante.

Se supone que un problema como éste tiene un equivalente en el mundo real y que, por lo tanto, posee un significado para el alumno.

En la práctica, por supuesto, el grado de éxito que alcancen estos objetivos dependerá de la disponibilidad de hardware y software de la escuela. Aun así, el proyecto, como ejercicio para la resolución de problemas de la vida real, es (desde el punto de vista educativo) uno de los argumentos de mayor peso para la inclusión de un curso de informática en las escuelas secundarias.

El plan de estudios típico de informática a nivel avanzado amplía el del nivel básico en la profundización de la comprensión requerida más que en el contenido del curso. En cierta medida, sin embargo, es más difícil justificar el examen a este nivel.

En general las universidades no requieren informática a nivel avanzado como prerrequisito para otorgar un título en esta disciplina, lo que da lugar a la suposición de que tal curso

empieza desde un grado elemental. De modo similar, muchos cursos de ordenador que se imparten en establecimientos de educación superior no dan por sentado ningún conocimiento previo. Es difícil, entonces, discernir con exactitud dónde queda situado el nivel del curso avanzado. Una posibilidad es que las escuelas de educación superior eleven sus expectativas y, como en el caso de otras asignaturas, exijan a los estudiantes algunas cualificaciones previas para acceder a los cursos de informática. A la larga, ello seguramente elevaría los niveles finales de educación.

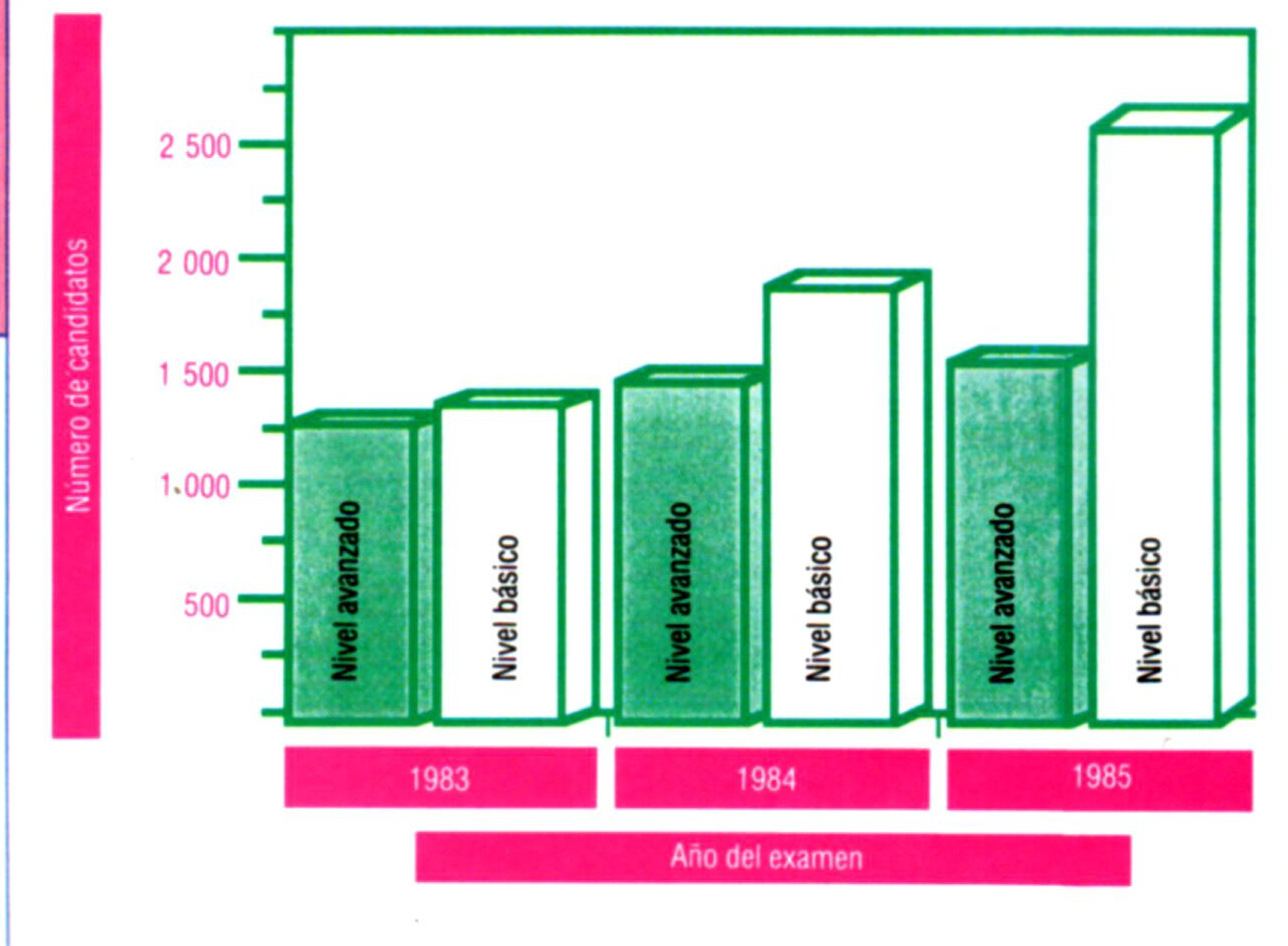
Aparte de los cursos para exámenes estándares, muchas escuelas británicas han introducido cursos de lo que se ha dado en llamar "apreciación de ordenadores" o "tecnología de la informática" (TI). Estos cursos se han creado atendiendo a diversos motivos. En muchos casos, las escuelas comprenden que existe un "vacío en el plan de estudios" entre el creciente uso de los ordenadores por parte de los alumnos de escuelas primarias-superiores y los cursos de las escuelas secundarias. Otros centros de enseñanza reaccionan ante las presiones de los padres para incluir cursos sobre temas que los padres consideran como "el conocimiento del futuro". Además, los cursos desarrollados recientemente CPVE y 16-Plus para alumnos de nivel no avanzado también exigen un componente de "tecnología de la información". El grado con el cual se abordan estas innovaciones (incluso donde se las reconoce) varía enormemente en la práctica. La razón de ser de un curso de esta naturaleza a menudo es bastante confusa, con el resultado de que los alumnos acaban "jugando con el ordenador".

Tecnología de la información

Idealmente, los objetivos de un curso de TI son proporcionar una visión profunda de métodos de base tecnológica para reunir, almacenar, procesar y distribuir información en todas sus formas (vocal, textual, iconográfica, numérica, etc.) Centrándose en métodos en los que se pueda utilizar un ordenador, se persigue que todos los alumnos tengan confianza en el uso del ordenador para reunir, almacenar y acceder a la información. En el proceso didáctico se suelen emplear, por ejemplo, paquetes de recuperación de información, procesadores de textos o sistemas de videotex.

Es evidente que las aptitudes requeridas son importantes para todo el aprendizaje; ya es opinión generalizada que un curso de tecnología de la información en los primeros años de la escuela secundaria es tan esencial como estudiar lenguaje y literatura o matemáticas. A medida que las escuelas dispongan de paquetes de software eficaces, no sólo aumentará la gama de las actividades posibles, sino que también se hará menos hincapié en la enseñanza de programación, arquitectura del ordenador, etc. A consecuencia de estos factores, bien puede ser que desaparezca la informática como materia de examen. Esto podría abrir las puertas a la posibilidad de cursos más específicamente vocacionales, como tratamiento de textos, etcétera. Por último, quizá veamos a los ordenadores en las escuelas como un "departamento de servicios", utilizado indistintamente por la administración, los maestros y los alumnos, lo que posiblemente redundaría en ventajas para todos.

Estudios de informática en Gran Bretaña
Estas cifras, dadas a conocer por el London School Examination Board, muestra el sorprendente crecimiento del número de alumnos que cursan informática en nivel básico. Por el contrario, el aumento en la cantidad de estudiantes que se presentan a exámenes del nivel avanzado ha sido mucho menos acusado. Con esto cobra fuerza la hipótesis de que, al terminar la escuela, los estudiantes ven el conocimiento básico de la informática como un valioso complemento de otros estudios, pero no necesariamente deseable como objeto de una especialización



Caroline Clayton

Intento unificador

Plan unificado de estudios de informática en un establecimiento educacional de Londres:

Aptitudes para la comunicación
— Representación e interpretación de algoritmos
— Necesidad y contenido de una documentación adecuada

Informática básica
— Hardware (CPU, memoria, características funcionales de dispositivos de entrada y salida y almacenamiento de apoyo)
— Arquitectura del ordenador (registros, ciclo buscar-ejecutar)
— Representación para la máquina (enteros, caracteres, instrucciones)
— Software (lenguajes de alto y bajo nivel, traductores, errores y ayudas para el diagnóstico, sistemas operativos, paquetes de aplicaciones)

Proceso de la información
— Principales aspectos del análisis de sistemas
— Captura de datos, validación, transmisión y validación
— Métodos para reducir los errores
— Especificación y diseño de archivos
— Especificación de documentos de E/S
— Descripción de aplicaciones utilizando sistemas de diagramas de flujo
— Métodos para la recuperación y seguridad de archivos
— Métodos para el proceso (por lotes, tiempo real, distribuido, etc.)
— Limitaciones e idoneidad del uso del ordenador en aplicaciones dadas
— Efectos sociales tales como intimidad y acceso a los datos

Resolución de problemas mediante el uso de un ordenador
— Capacidad para describir un problema, captar los datos, usar un ordenador para procesar estos datos y presentar los resultados



Función formal



Examinaremos la estructura y el papel de las funciones en C y analizaremos un programa de ejemplo

El lenguaje C se basa fundamentalmente en el concepto de *función*. No es un lenguaje funcional como tal, porque la forma en que opera cada función se define de modo procedural. No obstante, en C virtualmente todo se define desde el punto de vista del concepto *función*.

Lo que queremos significar por *función* en este contexto es un proceso al que se le pasan ciertos valores (los argumentos o parámetros) y que utiliza esos valores para producir un único valor que es devuelto. Los valores no han de ser necesariamente tipos básicos, como enteros o reales, sino que pueden ser tipos estructurados tales como matrices.

Hay otros varios conceptos esenciales que debemos considerar: los bloques, por ejemplo. En C, un *bloque* es una sección autocontenida de código, indicada mediante llaves ({ }); en PASCAL, a fines de comparación, un bloque se encierra entre sentencias begin...end. Asimismo, necesitamos valorar la idea del *ámbito*, es decir, la región en la cual está disponible para utilizarse. Son pocas las versiones de BASIC que admiten variables locales, de modo que es posible que muchos programadores de BASIC no estén habituados a la idea de tener que restringir el uso de ciertas variables a ciertas porciones de un programa.

En C, las variables se pueden declarar a lo largo de un programa y, en consecuencia, es especialmente importante saber cuándo y dónde se puede aludir a ellas.

Las reglas básicas para la declaración de variables en C son:

- Cuando se declara una variable al comienzo de una definición de función, su ámbito es la totalidad del cuerpo de función, incluyendo cualquier otra función que pudiera definirse dentro del cuerpo de la función original. En particular, dado que el programa se compone de la función main(), las variables declaradas al comienzo tendrán como ámbito la totalidad del programa, siendo, por tanto, variables *globales*.

  En el diseño de programas, es un aspecto de creciente importancia el que tales módulos de un programa sean totalmente autocontenidos y utilicen sólo variables declaradas localmente y parámetros formales. Si la ejecución de una función implica otros datos aparte de los de las variables locales, entonces se dice que esto es un *efecto lateral*. Por ejemplo, la operación de un dispositivo de entrada/salida por lo general es un efecto lateral, como lo es el uso de toda variable global. Algunos efectos laterales son beneficiosos, es decir, no tienen efecto alguno sobre la ejecución de otras partes del programa; pero otros son nocivos y su uso se debe evitar siempre que sea posible. Esto se aplica especialmente al uso de variables globales.
- Cuando se declara una variable dentro de un bloque, su ámbito se extiende solamente al resto de ese bloque.

La declaración de variables es especialmente importante porque, como veremos más adelante, forma parte de la "filosofía" del C el que un programa se haya de construir en módulos. Cada uno de estos módulos suele existir en un archivo separado, y las reglas que gobiernan el hecho de que una variable pueda tener un ámbito que incluya funciones definidas en un archivo diferente son algo complejas. Más adelante analizaremos estas reglas.

Hay otras dos formas de pasar argumentos o parámetros a una función:

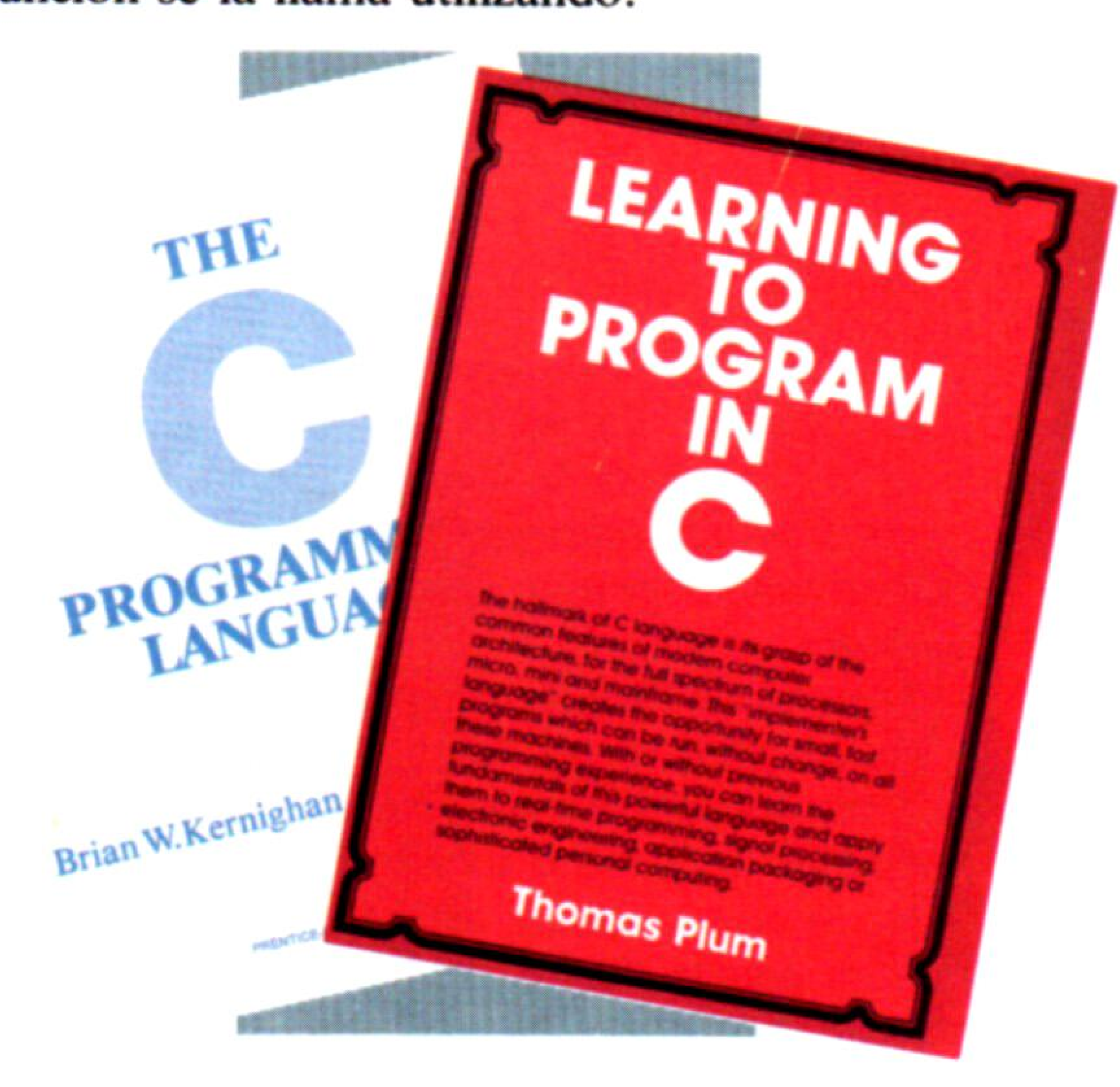
- *Valor:* Donde se crean variables enteramente nuevas. Estas variables son locales a la función y se inicializan en los valores pasados por la rutina de llamada.
- *Referencia:* Donde se pasa la dirección de la variable que contiene al parámetro. De este modo, se utiliza la misma variable en la función y la rutina de llamada.

En C, todos los parámetros se pasan por valor; pero, como veremos más adelante, el lenguaje posee amplias facilidades para manipular direcciones o punteros de variables y, por tanto, se puede obtener la llamada por referencia.

La sintaxis formal para una definición de función es:

```
tipo_función nombre_función (lista
   —argumentos_formales);
declaraciones_variables;
{
        cuerpo_de_la_función;
}
```

El tipo_función se puede omitir si el valor devuelto es int. La lista_argumentos_formales es una lista de variables de los tipos adecuados en las que se colocarán los valores cuando se utilice la función. A la función se la llama utilizando:



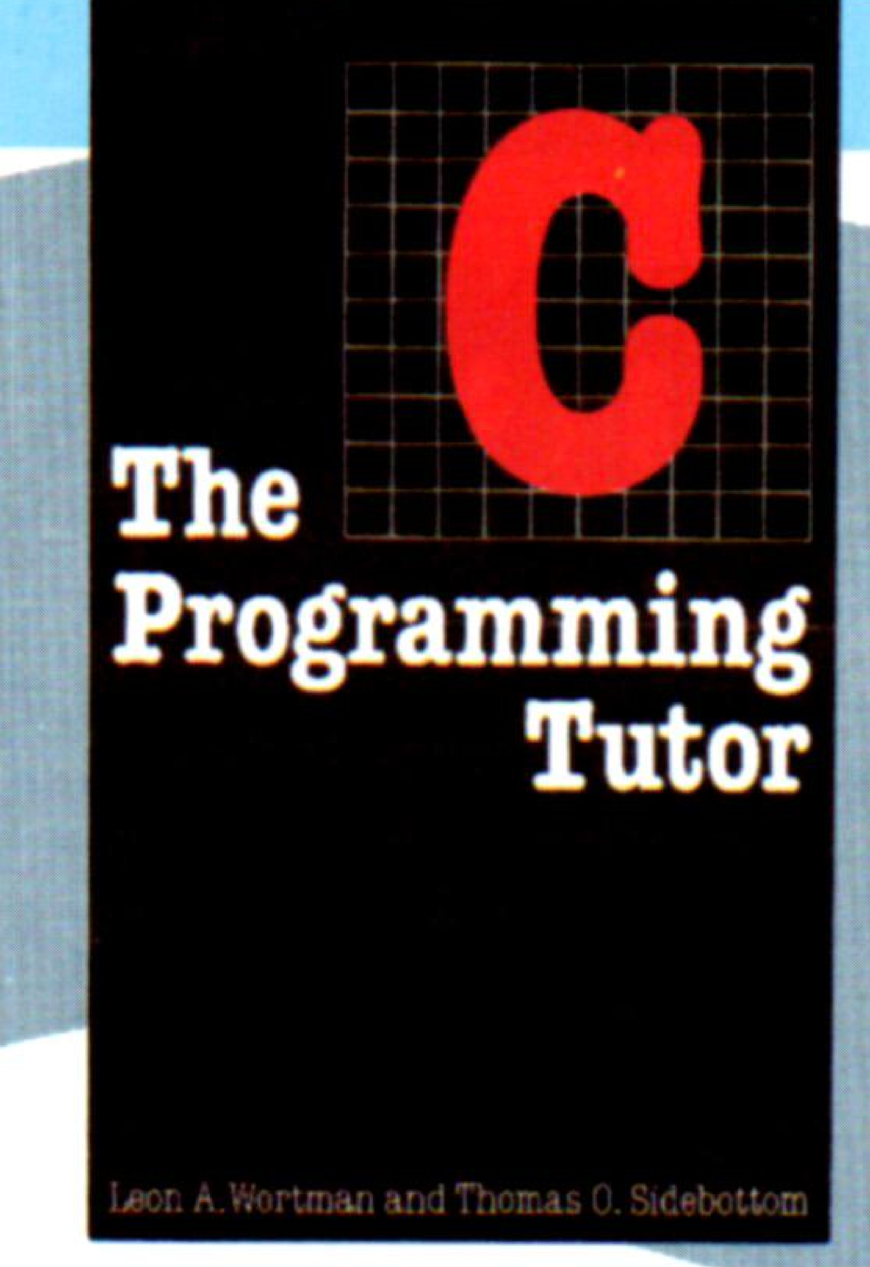
Biblioteca C
La editorial Prentice-Hall ha tenido una notable influencia en la creciente popularidad del C al mantener una amplia lista de publicaciones dedicadas al tema, incluyendo el texto *The C programming language* (El lenguaje de programacion C), original de Kernighan y Ritchie. Por su alto precio, este escueto volumen no está al alcance de muchos usuarios; no obstante, hasta que se autorice un estándar internacional, seguirá siendo la especificación definitiva del lenguaje. Otro texto importante editado por Prentice-Hall es *Learning to program in C* (Aprendiendo a programar en C), de Thomas Plum

```
nombre_función(lista_argumentos_actuales)
```

en cualquier punto donde sea adecuado utilizar un valor del tipo que devuelva la función. No obstante, si el tipo del valor devuelto es distinto de int, entonces el nombre de la función debe haberse declarado como una variable del tipo adecuado antes de poder llamarla. La lista de argumentos actuales es la lista de aquellos valores, o variables que contienen aquellos valores, que se han de pasar a la función y colocar en los argumentos formales. Los argumentos reales y formales, por supuesto, deben coincidir en tipo y posición en las dos listas.

El último detalle que debemos considerar antes de escribir una función es el uso de la sentencia return para dar el valor que se ha de devolver a la rutina de llamada. Puede haber una cantidad cualquiera de estas sentencias, pero se debe ejecutar una antes de salir de la función.

El siguiente programa entrará un conjunto de



Claro y conciso
The C programming tutor (El preceptor de programación en C), de Wortman y Sidebottom, constituye un excelente texto para principiantes. Escrito en un estilo claro y conciso, introduce y explica temas complejos sin confundir al lector y es especialmente recomendable para aquellos lectores que carezcan de experiencia en lenguajes de programación, exceptuando el BASIC. Este libro ha sido publicado por la editorial británica Prentice-Hall

números reales e imprimirá los miembros mayor y menor del conjunto. El programa utiliza tres funciones. Una de ellas, imprimircabecera, no tiene argumentos ni sentencia return, lo que significa que el valor devuelto por la función no está determinado. Puesto que la rutina de llamada no requiere el valor devuelto, éste simplemente se "descarta". Esta función sólo tiene efectos laterales, pero todos ellos son beneficiosos y, por lo tanto, no hemos de preocuparnos por ellos.

El programa también utiliza la función de biblioteca scanf, que realiza la entrada formateada de un modo muy similar a la empleada por printf para salida. El primer parámetro para la función es una serie que contiene información sobre el formato de los valores a entrar, utilizando la misma técnica que printf. Los otros argumentos de la función scanf son las variables en las cuales se han de colocar los valores. No obstante, dado que los parámetros sólo se pueden pasar por valor, no se los puede utilizar para devolver valores a la rutina de llamada. Por consiguiente, es necesario que estos parámetros sean direcciones de variables en vez de las variables propiamente dichas. Cada variable nombrada en la llamada a scanf debe ir precedida por el operador de dirección, &. Más adelante veremos otros usos de este operador.

```
# include<stdio.h>
main ( )
{
   int contador,tamaño_de_la_entrada,
   double máspequeño,másgrande,número,mín( ),
   máx( );
/*observe la declaración de las funciones mín y máx*/
/*observe también que la declaración dentro del
bloque hace que las variables no estén disponibles
fuera del bloque*/
   imprimircabecera( );
/*función llamada, se supone que el valor devuelto es
int y se lo "descarta", es decir, no es usado en ningún
sitio*/
   printf("\ntamaño de la entrada:");
   scanf("%d",&tamaño_de_la_entrada);
/*  averiguar cuántos números se van a entrar*/
   printf("Ahora entre %dnúmeros reales:\n",
   tamaño_de_la_entrada);
```



```
/*  tomar el primer número que será el másgrande y
máspequeño actual*/
   scanf("%lf",&número);
   másgrande=máspequeño=número;
/*  ahora tomar los otros números*/
   for(contador=2;contador<=tamaño__de__la__
      entrada;
      ++contador)
      {
         scanf("%lf",&número);
         máspequeño=mín(máspequeño,número);
         másgrande=máx(másgrande,número);
      }
      printf("\nmáspequeño es &f\nmásgrande es
      %f\n",máspequeño,másgrande);
}
/*  ahora vienen las definiciones de funciones*/
imprimircabecera ( )
/*  no se necesita tipo, se supone que es int*/
{
   printf("\n%s\n%s\n%s\n",
      "Entre primero el tamaño del conjunto de
            números",
      "después entre esa cantidad de números
            reales.»,
      "Se visualizarán el más grande y el más
            pequeño");
}
double mín(x,y);
double x,y;
/*  observe que se declaran todos los parámetros*/
{
   if(x<y)
      return(x);
   else
      return(y);
}
double máx(x,y);
double x,y;
{
   if(x>y)
      return(x);
   else
      return(y);
}
```

El preprocesador c

Muchos compiladores, en diversos lenguajes, configuran *directivas de compilación* incorporadas en un programa. Éstas son ciertas líneas que se reconocen como instrucciones dirigidas al compilador y no como sentencias del lenguaje. El c lleva esta idea un paso más hacia adelante que los otros lenguajes, y todos los compiladores de c incorporan un *preprocesador* que reconoce ciertas *líneas de control* como instrucciones para alterar consiguientemente el programa antes de presentárselo al propio compilador. Las líneas de control se distinguen mediante un signo # al comienzo. Asimismo, las directivas del preprocesador no necesitan ir seguidas de punto y coma, puesto que no se consideran como parte del programa.

La directiva del preprocesador # include <nombrearchivo> o # include"nombrearchivo" indica al preprocesador que incluya en este punto el contenido del archivo mencionado. Puede que en algunos sistemas no exista diferencia entre el uso de paréntesis en ángulo, <>, y comillas; pero otros sistemas pueden hacer una diferencia en los lugares que se buscan con el fin de hallar los archivos. La convención habitual es que los paréntesis en ángulo se emplean para archivos del sistema, mientras que las comillas se utilizan para los propios archivos del usuario. Las funciones printf y scanf que hemos estado empleando están incluidas en un archivo estándar del sistema que se denomina stdio.h, de modo que todos los programas que las utilicen deben incluir la directiva # include<stdio.h>, como al comienzo de nuestro programa.

Se pueden anidar estas llamadas de modo que uno de los archivos que se incluya posea también un # include para incorporar otro archivo.

La directiva # define permite una forma limitada de macrosustitución. En su forma más simple, permite la definición de una constante; por ejemplo, # define EOF-1 haría que el identificador EOF se reemplazara por –1 cada vez que apareciera en el archivo del programa. Esta facilidad hace que resulte mucho más sencillo realizar cambios en valores "constantes" en el caso de que surja la necesidad. Otros ejemplos son:

```
# define PI3.14159
# define EQ==
```

La convención generalmente aceptada es que los identificadores definidos de este modo se escriben en mayúsculas, reservando las minúsculas para las variables normales. Se pueden incluir parámetros en una macrodefinición, como en

```
# define SUMSQ(x,y)((x * x)+(y * y))
```

que tiene un efecto similar a las definiciones de función en BASIC. Por ejemplo, la ocurrencia de un SUMSQ(3,4) se reemplazaría por ((3 * 3)+(4 * 4)). Observe la inclusión de paréntesis como medida de seguridad; la macro se podría utilizar en una situación en la que los paréntesis fueran necesarios, lo que significa que siempre es mejor incluirlos.

Existe una directiva # undef que hace que el preprocesador no realice más sustituciones para la macro mencionada después de haberla encontrado.

Otra de las directivas comúnmente utilizadas es la construcción # if... # else... # endif, que permite la compilación condicionada. Si la expresión de constante que sigue a la sentencia # if se evalúa como verdadera (no cero), entonces se compilan las líneas de código que siguen tras la sentencia. Si la expresión de constante se evalúa como falsa (cero), entonces se compilan las líneas que siguen al # else.

Un ejemplo de su utilización se puede observar durante el desarrollo de un programa, cuando puede ser de ayuda incluir muchísimas sentencias printf adicionales al objeto de llevar el registro de lo que está sucediendo. Lo que podemos hacer es colocar directivas de la forma # define DEBUG 1, y siempre que utilicemos un printf adicional colocamos:

```
# if DEBUG
      printf(valores...)
# endif
```

Luego, cuando hemos acabado la depuración, tan sólo necesitamos cambiar las directivas por # define DEBUG 0, y al volver a compilar nuestro programa éste ya no incluirá esas sentencias.

Oro macizo

Examinemos de cerca el que se considera el "Rolls Royce" del tratamiento de textos

Como casi todos los paquetes para tratamiento de textos muy caros, el *Samna Word III* se ejecuta en la gama de ordenadores IBM PC, siempre que operen bajo la versión 2.0 de MS-DOS o superior. Ello se debe a que el *Word III* implementa directorios de archivos y "vías" que sólo se introdujeron en el DOS 2.0. El paquete se compone de un manual de introducción, trazados del teclado Samna para varios lenguajes diferentes, un manual más detallado que explica cada una de las instrucciones, y cinco discos flexibles que contienen el programa propiamente dicho.

Los discos contienen varias utilidades diferentes. El programa principal está retenido en dos de los discos y otros discos retienen las rutinas de la impresora y el diccionario. El quinto, que en realidad no forma parte del programa principal, es el disco de entrenamiento, que contiene numerosos programas de enseñanza cortos que guían al usuario a través del programa.

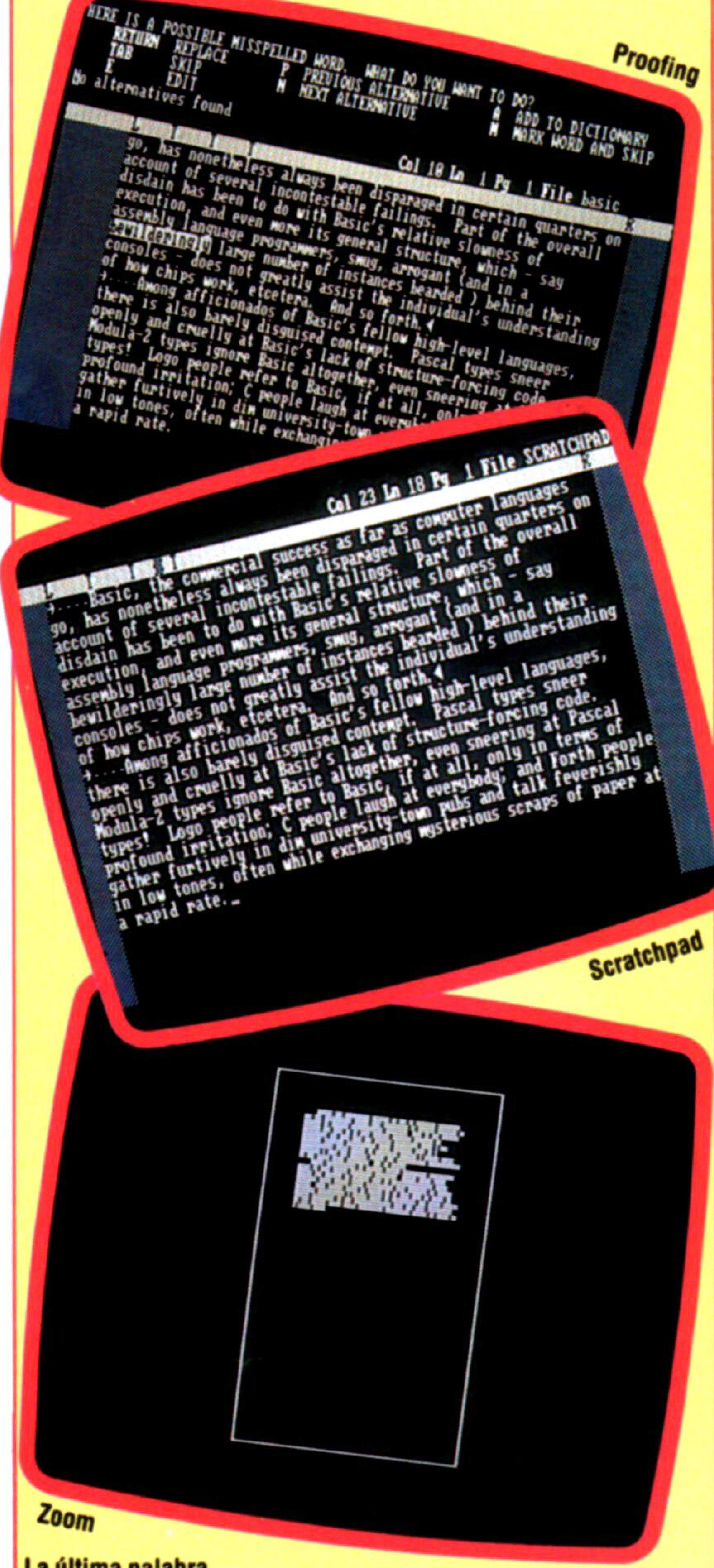
Una vez instalado en varios discos, el programa total ocupa más de 700 Kbytes de espacio de disco en 45 archivos del usuario: una cantidad considerable tratándose de un solo programa. La ejecución del programa presenta al usuario una pantalla Scratchpad, que informa que actualmente no hay cargado ningún archivo con nombre. A los lados de la pantalla hay barras coloreadas indicando las posiciones de los márgenes, y arriba de éstas otra barra que muestra los ajustes de tabulación y márgenes por defecto. Arriba de la pantalla hay indicadores que muestran la posición actual del cursor, el nombre del archivo y un aviso READY!.

Una de las grandes ventajas del IBM PC y sus compatibles es la gran cantidad de teclas programables que se han puesto a disposición de quienes desarrollan software. Los programadores del *Word III* han sacado de ello el máximo partido y no sólo han empleado las 10 teclas de función para proporcionar una serie de utilidades para tratamiento de textos, sino que también han reprogramado algunas de las otras teclas con la misma finalidad.

El teclado numérico, por ejemplo, se emplea como un grupo de cursor (una función utilizada normalmente por el teclado IBM), pero algunas de las teclas, como 7, 9 y 1, se han utilizado para desplazar el cursor a lo largo de una palabra, frase y párrafo, respectivamente. Por otra parte, la tecla Scroll Lock se ha reprogramado como la tecla Mark, que se utiliza para definir bloques que se pueden manipular dentro del texto. Las teclas de función se han programado para realizar las funciones más comúnmente utilizadas en tratamiento de textos, tales como el subrayado y el ajuste.

El *Word III*, sin embargo, es mucho más amplio que los otros programas que hemos analizado en esta serie. La llamada a una función, por ejemplo, a menudo conduce a otra pregunta que, a su vez, solicita al usuario exactamente sobre qué zona del texto ha de operar dicha función, como la totalidad del documento, el párrafo o todo cuanto haya después de la posición actual del cursor.

Los programadores de Samna han intentado conseguir los objetivos, con frecuencia incompatibles, de máximas flexibilidad y sencillez de uso. A modo de ejemplo, el *Word III* no posee facilidad de inserción automática. Cuando el usuario desee insertar texto en medio de un documento, pulsando



La última palabra
El *Samna Word III* es un sofisticado programa para tratamiento de textos que soporta facilidades de las que carecen los sistemas más económicos. Aparte de las funciones usuales de inserción y manipulación del texto que ofrecen la mayoría de los programas, el *Word III* posee complejas (aunque amables) facilidades para verificación de ortografía, correspondencia automática y formateo del texto. *Word III* posee una facilidad denominada Zoom, que da al usuario una idea del aspecto final de la página

SAMNA WORD III

Desplazamiento de palabras

Proporciona desplazamiento automático y ajuste de línea completa.

Movimiento de bloques

Tiene todas las instrucciones estándares, incluyendo el movimiento de columnas.

Ayuda en pantalla

El programa es uno de los más amables que hay disponibles desde este punto de vista.

Pantalla de 80 columnas

La pantalla por defecto establece los márgenes en 72 columnas, pero se puede alterar para usar las 80 columnas completas.

Contador de palabras
No tiene.

Buscar/reemplazar

Se pueden utilizar hasta 30 caracteres, con numerosas opciones de búsqueda.

WYSIWYG

Aunque el *Word III* posee facilidades muy útiles, tales como el Zoom, le faltan otras corrientes (como visualizar el subrayado en pantalla).

Facil. para correspondencia

Permite almacenamiento de bases de datos y correspondencia automática.

Verificador de ortografía

La utilidad Proof es rápida y amable, aunque quizá los usuarios británicos se impacienten a causa del diccionario norteamericanizado.

Tipos de letra disponibles

Si bien el *Word III* soporta subíndices y negritas, no dispone de tipos alternativos.

Unión de archivos
No tiene.

la tecla de inserción se visualizará un espacio adecuado. En el *Tasword II*, esta técnica requiere que el usuario vuelva a alinear el párrafo después de insertar el texto. En el *Word III*, sin embargo, con tan sólo volver a pulsar la tecla de inserción el cursor se libera de su posición y vuelve a alinear automáticamente el texto, con lo que se obtiene la mejor combinación.

El *Word III* posee una facilidad de ayuda sumamente útil. El usuario puede llamarla pulsando la tecla Help situada en el ángulo superior izquierdo del teclado de máquina de escribir, que visualiza la gama de opciones que hay disponibles. Por el contrario, si el usuario duda antes de llamar a una función, la pantalla de ayuda se visualizará a sí misma de forma automática.

Varias de las teclas de función están programadas para llevar a cabo una amplia gama de operaciones diversas. Entre las más útiles está la tecla Do. Entre las utilidades que se pueden entrar a partir de ella figuran muchas de las funciones de archivo normales, como borrar y copiar, pero la tecla también se utiliza para acceder a varias operaciones de las que carecen los procesadores de textos más económicos.

Cuando se escribe un documento largo, la pantalla sólo es capaz de visualizar una ventana. No obstante, si usted quisiera ver qué aspecto tendrá el documento entero en una página, la selección de la función Zoom visualizará el trazado de una página con barras mostrando dónde están posicionados los títulos y los párrafos. Esto le permite producir un trazado más atractivo sin tener que imprimirlo varias veces.

Otra de las funciones de la tecla Do supera otra limitación del efecto de ventana. Cuando usted está digitando numerosas columnas en un documento muy ancho, por ejemplo, quizá sea necesario aludir a una columna que no esté visualizada en la pantalla. La instrucción Fold "plegará" el documento automáticamente de modo que en la misma pantalla se puedan visualizar las dos caras de la página.

En la actualidad, uno de los requisitos de los paquetes de gestión es que sean compatibles no sólo con el IBM PC, sino también con otro software, en especial con el tan utilizado *Lotus 1-2-3*. Con el objeto de mantener la compatibilidad con este último, el *Word III* puede traducir archivos ASCII desde disco y visualizarlos en formato Samna. De este modo, dentro del *Word III* se pueden visualizar y manipular documentos del *Lotus*.

Todo procesador de textos que aspire a un lugar de privilegio en el mercado de gestión debe contar con facilidad para correspondencia automática y verificación de ortografía, estando ambas implementadas en el *Word III*. El verificador de ortografía, conocido como la instrucción Proof, es uno de los más amplios existentes actualmente en todos los paquetes.

Al igual que algunas utilidades del programa, el verificador de ortografía permite escoger entre numerosas opciones, especificando qué es lo que se desea "corregir". Una vez decidido esto, el ordenador revisa entonces cada palabra, cotejándola con las que tiene retenidas en su propio diccionario. Cuando encuentra una palabra que no reconoce, el ordenador visualiza una serie de opciones, incluyendo numerosas ortografías alternativas.

La utilidad de correspondencia automática implementada en el *Word III* se denomina Automatic Merge, y consiste en una pequeña base de datos en la que usted puede almacenar un cierto número de campos, tales como nombre, apellido y dirección. Éstos se pueden luego disponer en el sitio adecuado en un documento estándar e imprimir. Asimismo, el programa admite *máscaras* para buscar un determinado registro.

El *Word III* de Samna es un paquete para tratamiento de textos muy potente que resiste muy bien la comparación con numerosas máquinas exclusivas para tratamiento de textos. No obstante, sigue vigente la pregunta de si el paquete vale su alto precio. Si su empresa requiere un paquete para tratamiento de textos tan amplio, con facilidades que no encontrará en ningún otro, indudablemente la respuesta es afirmativa.

La letra pequeña

En el extremo opuesto de la escala que ocupa *Samna Word III* está *Mini Office*, de Database Publications. Este último no sólo incluye un programa para tratamiento de textos sino también utilidades de base de datos, gráficos y hoja electrónica. Se ejecuta en el BBC Micro, Electron, Commodore 64 y gama Amstrad. Por supuesto, el paquete para tratamiento de textos posee muy pocas de las sofisticadas funciones del *Word III* y, así y todo, el programa contiene numerosas facilidades de las que carecen programas más caros. Aunque la facilidad para tratamiento de textos en el *Mini Office* es básica, muchos usuarios que no requieran complejas funciones para formateo de la impresión encontrarán que el programa contiene todo cuanto necesitan para escribir cartas y otros documentos. El programa es activado por menú, que permite que el usuario seleccione varias opciones, a través de las teclas de función, para manipular el texto. Entre éstas se incluyen facilidades tales como cargar, guardar e imprimir documentos. La pantalla de textos soporta inserción/supresión y desplazamiento automáticos, si bien el programa no permite el ajuste de líneas. La parte superior de la pantalla proporciona información útil, incluyendo un contador de palabras, la cantidad de caracteres disponibles restantes y un reloj que le indica cuánto tiempo lleva digitando. Asimismo, el programa permite transcribir texto de una a otra parte de un documento con sólo copiar caracteres a partir de una posición del cursor previamente definida y hasta la posición actual

Liz Heaney

Defensa antiaérea

Este juego, también conocido como "D.C.A. (Defensa contra aviones)", invierte los papeles asignados en "Bombardeo aéreo". Ha sido escrito para el Commodore 64



Su misión es manejar las defensas contra aviones e intentar abatir los aviones que vuelan sobre su posición. Para disparar hay que usar una tecla cualquiera. Inicialmente dispone de quince obuses. Si consigue abatir diez aviones, obtendrá una bonificación de diez puntos y diez nuevos obuses adicionales.

```
5 REM **********************************
10 REM *              D.C.A             *
15 REM **********************************
20 GOSUB 1000
30 GOTO 720
100 A$=RIGHT$(A$,39)+LEFT$(A$,1)
110 B$=RIGHT$(B$,1)+LEFT$(B$,39)
120 PRINT A$
140 PRINT B$;HH$;
200 GET X$
210 IF X$<>" " AND M=MI THEN
    M=MI-80:NM=NM-1:GOTO 230
220 IF M<>MI THEN M=M-80
230 IF M<>M2+80 THEN 250
240 IF PEEK(M2-1)<>32 THEN 500
250 IF M<>M1+80 THEN 270
260 IF PEEK(M1+1)<>32 THEN 600
270 IF M=MI THEN 310
280 POKE M,CM
290 POKE M+N,MC
300 IF M+80<>MI AND M<>M1 THEN POKE
    M+80,CR
310 IF M<1064 THEN M=MI:GOTO 720
320 IF NM<1 AND M=MI THEN 4000
330 GOTO 100
500 B$=LEFT$(B$,17)+01$+RIGHT$(B$,16)
550 GOTO 650
600 A$=LEFT$(A$,17)+01$+RIGHT$(A$,16)
650 POKE M-80,160
660 POKE (M-80)+N,2
670 POKE M+80,CR
680 S=S+1
700 IF S>1 AND INT(S/10)=S/10 THEN GOSUB
    800
710 M=MI
720 FOR I=1 TO 20
730 PRINT CHR$(17);
740 NEXT I
750 PRINT "PUNTOS[1SPC]:";S
760 PRINT "OBUS[2SPC):";STRS$(NM);
    "[1SPC]"
770 PRINT HH$;
780 GOTO 310
800 A$=A1$:B$=B1$:NM=NM+10
830 FOR I=1 TO 500:NEXT I
850 S=S+10
860 RETURN
1000 A$=" ":B$=" ":HH$=CHR$(19)
1030 M1=1044:M2=1124:MI=2004
1060 M=MI:CM=30:MC=5:CR=32:
     NM=15
1110 01$=" ":N=54272:X$=" ":S=0
1200 FOR I=1 TO 7
1210 01$=01$+"[1SPC]"
1220 NEXT I
1230 FOR I=1 TO 40
1240 READ A,B
1250 A$=A$+CHR$(A)
1260 B$=B$+CHR$(B)
1270 IF I/8=INT(I/8) THEN RESTORE
1280 NEXT I
1290 A1$=A$:B1$=B$
1500 PRINT CHR$(147);
2000 FOR I=M-1 TO M+1
2010 POKE I,98:POKE I+N,5
2030 NEXT I
2040 POKE M,160
2050 POKE 53280,0
2060 POKE 53281,6
2070 PRINT CHR$(158);
2080 RETURN
4000 PRINT CHR$(147)
4010 IF S>RE THEN RE=S
4020 FOR I=1 TO 4
4030 PRINT
4040 NEXT I
4050 PRINT TAB(13)"PUNTOS[1SPC]:";S
4060 FOR I=1 TO 4
4070 PRINT
4080 NEXT I
4090 PRINT TAB(11)"PUNTUACION[1SPC]
     MAXIMA[1SPC]:";RE
4100 FOR I=1 TO 4
4110 GET X$
4120 PRINT
4130 NEXT I
4140 PRINT TAB(13)"OTRA[1SPC]?"
4150 GET X$
4160 IF X$=" " THEN 4150
4170 IF X$<>"N" THEN 20
4180 END
10000 DATA 32,182,162,162,162,162,181
10010 DATA 32,32,32,32,32,32,32,32,32
```



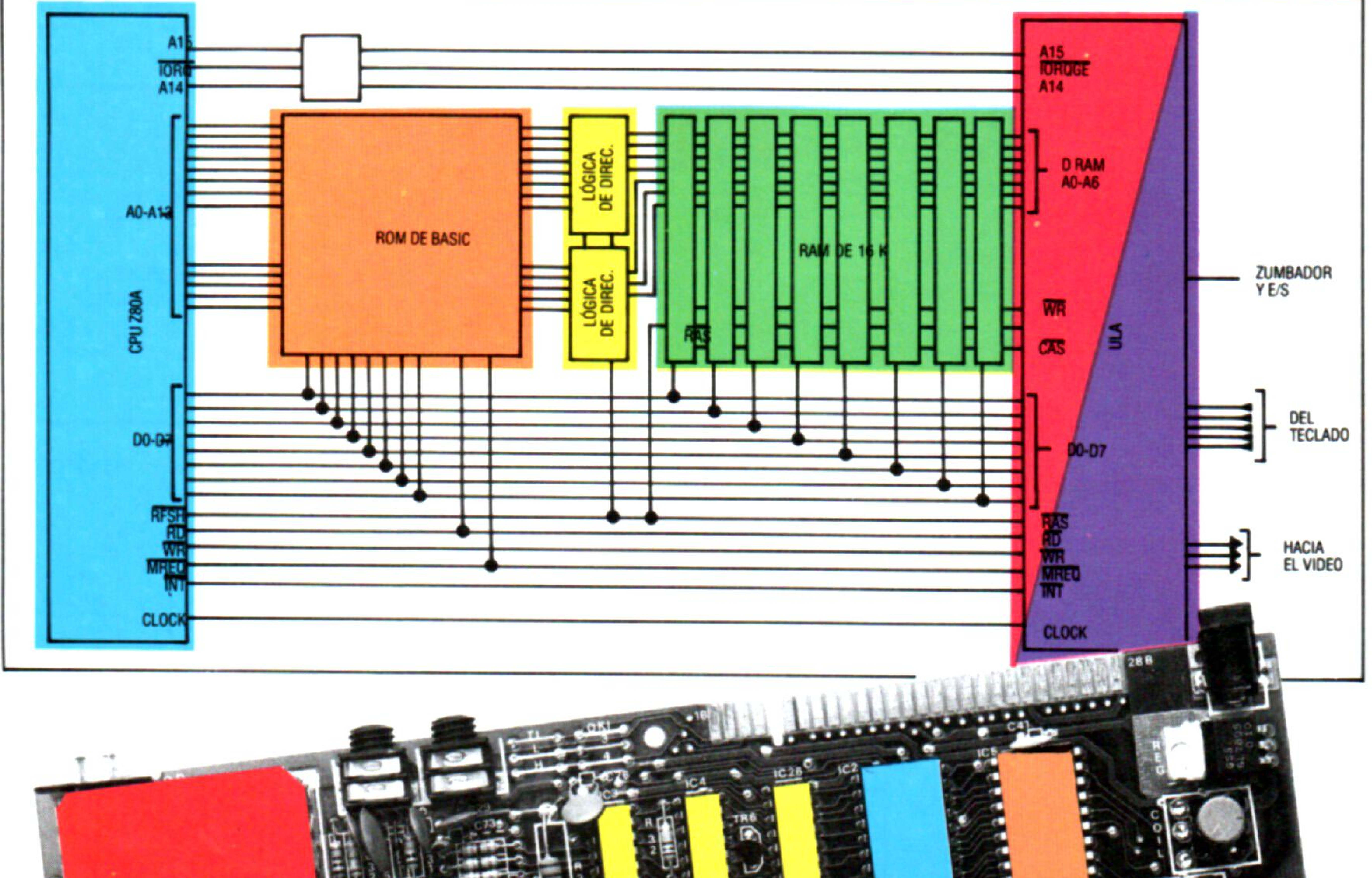
Interiores

Ponemos punto final a esta serie dedicada a analizar el hardware de los ordenadores examinando el interior de tres populares micros: Sinclair Spectrum, Commodore 64 y BBC Modelo B. De esta manera podremos identificar en cada máquina los componentes que hemos estudiado. Asimismo, proporcionamos diagramas de lógica simplificada para demostrar cómo se concretan los circuitos teóricos.

Sinclair Spectrum

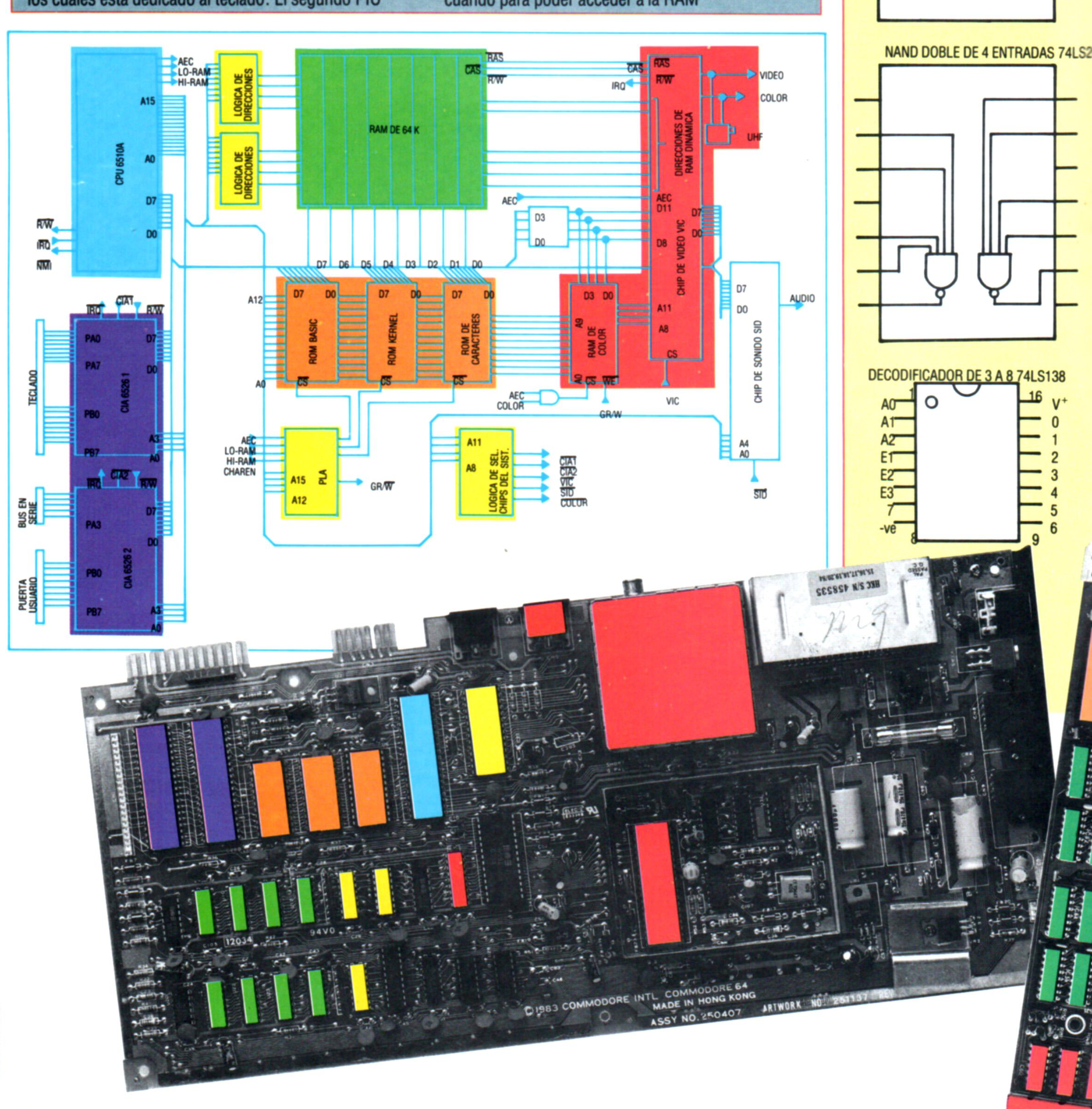
Desde el punto de vista de su arquitectura, el ZX Spectrum es bastante distinto de las otras máquinas que vemos aquí. Basado en el procesador Z80, la mayoría de las funciones del Spectrum las lleva a cabo una ULA especial. Este chip diseñado a medida se ocupa del proceso de video, refresco de memoria y E/S mínima desde teclado, grabadora de cassette y zumbador. El diagrama (abajo) corresponde a uno de los primeros Spectrum de 16 K. Las versiones de 48 K poseen chips de RAM extras y decodificación de direcciones. Al igual que en todos los sistemas de visualización de video para micros, la ULA necesita realizar accesos rápidos y regulares a la memoria. En el Spectrum de 48 K, el primer bloque de 16 K, que contiene los datos de video, está conectado directamente a la ULA. En circunstancias normales, ésta puede acceder a esta área de 16 K independientemente, mientras la CPU accede a la ROM de BASIC o al área de RAM extra de 32 K. El problema se plantea cuando la CPU desea acceder al área de video de 16 K (cosa que puede hacer, porque allí están almacenadas las variables del sistema de BASIC). Para evitar el conflicto entre la CPU y la ULA, la ULA monitoriza las líneas de dirección A14 y A15 y detiene temporalmente el reloj de la CPU si se produce un conflicto de acceso





Commodore 64

Desde el punto de vista del hardware, el Commodore 64 es una máquina bien equipada. Posee 64 K de RAM dinámica, proporcionada por ocho chips de RAM de 64 Kbits, y tres ROM de ocho K que contienen el BASIC, las rutinas kernel de E/S y las definiciones de caracteres. El procesador 6510 es una versión mejorada del 6502 que permite conmutar las ROM en el espacio de direcciones del procesador a través de un registro y una PLA especial del chip. El C64 tiene dos chips PIO, uno de los cuales está dedicado al teclado. El segundo PIO utiliza una puerta para controlar el propio sistema de bus en serie de Commodore, al cual se pueden conectar impresoras, unidades de disco y otros dispositivos periféricos; la otra puerta queda libre como puerta para el usuario. El chip VIC direcciona la RAM dinámica directamente para obtener información de pantalla. Puesto que necesita más tiempo de comunicación con la memoria que el que se puede conseguir realizando accesos durante la fase del reloj del sistema en que la CPU no está utilizando los buses de datos y direcciones, el VIC detiene temporalmente el procesador de cuando en cuando para poder acceder a la RAM



Deducciones lógic

Chris Stevens



NAND DE 8 ENTRADAS 74LS30

NOR CUAD DE 2 ENTRADAS 74LS02

Lógica aplicada
Muchos de los chips más pequeños que se encuentran en la placa impresa de un micro pertenecen a la familia 74** o 74LS**. Estos chips contienen diversas combinaciones de puertas lógicas y se utilizan para *gating* o decodificar señales de dirección o control. Aquí vemos los detalles de varios chips lógicos 74LS**

BBC Modelo B Micro

Diseñado originalmente en 1981, el BBC Modelo B todavía utiliza 16 chips de RAM dinámica de 16 K para proveerse de 32 K de memoria. Se utilizan seis chips de lógica de direccionamiento para permitir que la CPU y los chips de video y teletexto compartan la RAM. En la configuración básica se proporcionan dos ROM de 8 K así como tres ranuras laterales de ROM para ROMs adicionales.

Se proporcionan dos PIO, así como un chip de E/S en serie (que no se incluye en el diagrama). El primer PIO es exclusivo para el sistema y controla los chips del teclado, de sonido y de voz. Este PIO también monitoriza los botones de disparo de la palanca de mando y controla algunas de las funciones de video tales como de desplazamiento de pantalla. La CPU no se comunica directamente con los sistemas de teclado y sonido, sino que utiliza la puerta A del PIO del sistema como si fuera en realidad un bus de datos lento. El segundo PIO permite la conexión de una impresora en paralelo y a la puerta PIO que queda libre se puede acceder a través de la puerta para el usuario

Kevin Jones

Mano a mano

Prosiguiendo nuestro proyecto, completaremos la programación correspondiente al jugador y analizaremos la mano en preparación para el turno del ordenador

Ya hemos desarrollado una rutina con fines generales que sumará el total de una mano y establecerá una variable, EF, en diversos valores correspondientes a los posibles estados de la mano. Ahora podemos usar esta rutina para poder terminar la mano del jugador.

Recuerde que en esta etapa del juego se le habrán repartido al jugador y a la banca dos naipes a cada uno. El jugador puede plantarse o bien pedir carta para acercarse más a 21 sin pasarse. Para inclinar un poco más el juego a favor de la banca hemos incluido la regla que impide que usted se plante si su mano totaliza menos de 17. En un próximo capítulo incluiremos una tercera opción de apuesta que le permitirá doblar su apuesta y se le reparta un naipe más.

La línea 120 llama a la rutina pedir/plantarse del bucle principal del juego después de haber comprobado la opción quemar. La subrutina de la línea 2600 en realidad no realiza el trabajo de pedir carta o plantarse sino que, en cambio, llama a otra subrutina de la línea 2700 para que lleve a cabo la tarea. Al retornar de esta subrutina, usted habrá completado su mano y EF estará establecida en uno de los cinco estados posibles de la mano, como *royal pontoon* ("veintiuno real") o *bust* ("pasarse"). Median-

La mano del jugador

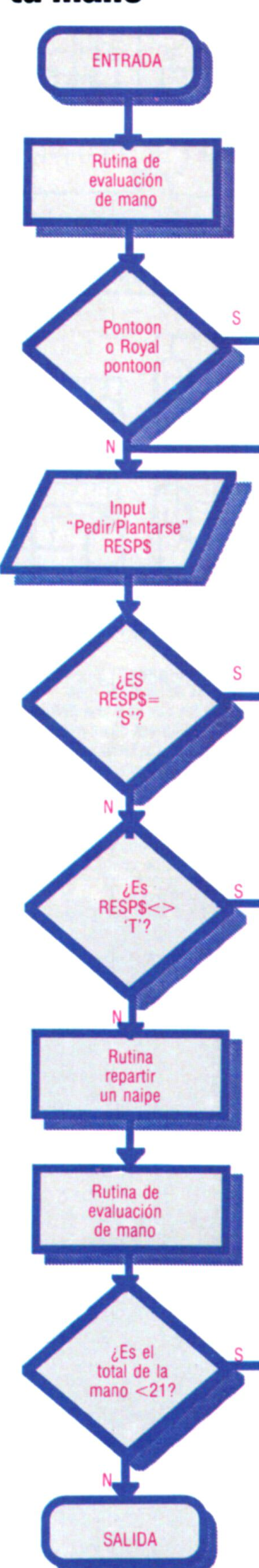
BBC Micro

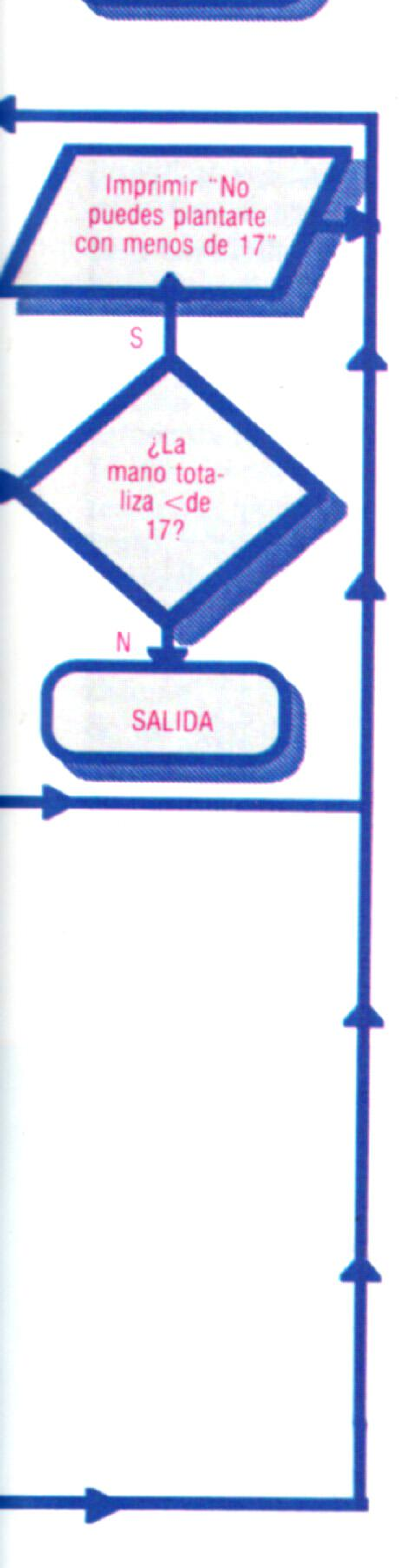
```
 120 GOSUB 2600
2600 REM
2610 GOSUB 2700
2620 ON EF GOSUB 3500,3600,3700,3800,3900
2630 RETURN
2700 REM
2710 GOSUB 800:IF EF=2 OR EF=3 THEN
     RETURN
2720 GOSUB 700:PRINT"PEDIR/PLANTARSE/
     DOBLAR";
2725 RESP$=GET$
2727 IF RESP$="S" THEN GOSUB 2800:IF CS=0 THEN
     RETURN
2730 IF RESP$="S" AND CS=1 THEN 2700
2750 IF RESP$<>"T" THEN 2700
2760 FL=0:PL=1:GOSUB 1300
2770 GOSUB 800:IF EF=1 THEN 2700
2780 RETURN
2800 REM
2810 CS=1:GOSUB 800
2820 IF (TT(PL,2)> 17 AND TT(PL,2)< 22) OR TT(PL,1)>=17
     THEN CS=0: RETURN
2825 GOSUB 700:PRINT"NO PUEDES PLANTARTE CON MENOS
     DE 17"
2830 FOR DL=1 TO 5000:NEXT DL
2840 RETURN
3500 REM **** MENOS DE 21 ****
3505 PV=1:PS=TT(1,2):IF PS> 21 THEN PS=TT(1,1)
3510 GOSUB 700:PRINT"MENOS DE 21":FOR DL=1 TO
     500:NEXT DL:RETURN
3600 REM **** ROYAL PONTOON ****
3605 PV=4
3610 GOSUB 700:PRINT"ROYAL PONTOON":RETURN
3700 REM
3705 PV=2
3710 GOSUB 700:PRINT"PONTOON":RETURN
3800 REM
3805 PV=0
3810 GOSUB 700:PRINT"BUST":RETURN
3900 REM
3905 PV=3
3910 GOSUB 700:PRINT"JUEGO DE CINCO NAIPES":
     RETURN
```

Gama Amstrad CPC

```
 120 GOSUB 2600:REM PEDIR etc.
2600 REM **** PEDIR apostador etc. ****
2610 GOSUB 2700:REM hacerlo!
2620 ON ef GOSUB 3500,3600,3700,3800,3900
2630 RETURN
2700 REM **** pedir/plantarse/doblar ****
2710 GOSUB 800:IF ef=2 OR ef=3 THEN RETURN:REM
     comprobar pontoon/royal pontoon
2720 GOSUB 700:PRINT" pedir/plantarse/doblar";
2725 resp$="":WHILE resp$"":resp$=INKEY$:
     WEND
2727 IF resp$<>CHR$(13) THEN PRINT resp$
2730 IF resp$="s" THEN GOSUB 2800:IF cs=0 THEN
     RETURN
2733 IF resp$="s" AND cs=1 THEN 2700:REM no se puede
     plantar
2750 IF resp$ <> "t" THEN 2700:REM error de
     entrada
2760 fl=0:pl=1:GOSUB 1300:REM repartir
2770 GOSUB 800:IF ef=1 THEN 2700:REM otra
     vez?
2780 RETURN
2800 REM **** plantarse ****
2810 cs=1:GOSUB 800:REM evaluar
2820 IF (tt(pl,2)>=17 AND tt(pl,2)<22) OR tt(pl,1)>=17 THEN
     cs=0:RETURN
2825 GOSUB 700:PEN rojo:PRINT "No te puedes plantar con
     menos de 17"
2830 FOR dl=1 TO 100:NEXT dl
2840 RETURN
3500 REM **** menos de 21 ****
3505 pv=1:ps=tt(1,2):IF ps>21 THEN ps=tt(1,1)
3510 GOSUB 700:PRINT "Menos de 21":RETURN
3600 REM **** royal pontoon ****
3605 pv=4
3610 GOSUB 700:PRINT "Royal pontoon":
     RETURN
3700 REM **** pontoon ****
3705 pv=2
3710 GOSUB 700:PRINT "Pontoon":RETURN
3800 REM **** bust ****
3805 pv=0
3810 GOSUB 700:PRINT "Bust": RETURN
3900 REM **** juego de cinco naipes ****
3905 pv=3
3910 GOSUB 700:PRINT "Juego de cinco naipes":
     RETURN
```

Muéstrame tu mano





El diagrama de flujo muestra cómo se controla la parte del juego correspondiente al jugador. La salida de la estructura de control sólo se puede conseguir bajo tres condiciones: si los dos naipes que se le repartieron originalmente al jugador dan *pontoon* (un as y un diez) o *royal pontoon* (un as y una figura), si el jugador se planta legalmente o si el jugador se pasa

te el uso de la instrucción ON...GOSUB (exceptuando la versión para el Spectrum, que saca partido de la facilidad de números de línea calculados del BASIC Spectrum), el valor de EF dirige al programa hasta otra subrutina. Ésta imprime un mensaje en el cual se le informa sobre el estado de la mano completada, y establece una variable PV. Esta variable se utilizará más adelante, cuando le toque el turno al ordenador, para retener el estado de la mano suya.

La auténtica rutina pedir/plantarse empieza en la línea 2700. Inmediatamente después de entrar en ella se llama a la rutina de evaluación. Se comprueban tanto el *royal pontoon* (un as y una figura) como el *pontoon* de dos naipes (un as y un diez) en los dos naipes que ya se han repartido al jugador. La rutina continúa pidiendo al jugador que entre T para pedir o S para plantarse y después comprueba el teclado en busca de una respuesta.

Utilizando GET/INKEY$/GET$ en lugar de INPUT, usted simplemente puede pulsar las teclas T o S sin tener también que pulsar la tecla Return (tener que pulsar ésta una y otra vez suele resultar molesto). La desventaja de usar GET/INKEY$/GET$ es que las pulsaciones de tecla no generan automáticamente las adecuadas visualizaciones en pantalla, como lo hacen con INPUT. Por consiguiente, necesitamos añadir líneas a nuestro programa para visualizarlas.

Si en este punto usted decide plantarse, el programa debe comprobar su mano para determinar si es o no menor que 17. La rutina de la línea 2800 comprueba esto mirando los marcadores de mano para el jugador, que ya habrán sido calculados por la rutina de evaluación de manos. Si su mano totaliza menos que 17, se imprime un mensaje. Además, se establece una bandera, CS, en 1. Al volver de la rutina pedir/plantarse, CS determinará si usted está o no en condiciones de plantarse, en cuyo caso se puede salir de la rutina. No obstante, si el programa le pide que vuelva a seleccionar, volverá hacia atrás en el bucle hasta el comienzo de la rutina.

Si selecciona la opción pedir, se le repartirá un naipe y se volverá a evaluar su mano. Si su mano suma aún menos de 21 (lo que se indicaría con EF=1) después de repartirse el nuevo naipe, la rutina vuelve hacia atrás en el bucle para una nueva entrada.

Con esto completamos la programación para la mano del jugador. En el próximo capítulo veremos cómo se puede programar el ordenador para que sea capaz de replicar "inteligentemente" a la mano del jugador.

Sinclair Spectrum

```
120 GO SUB 2600: REM PEDIR ETC.
2600 >REM **** PEDIR APOSTADOR ETC EVALUAR ****
2610 GO SUB 2700: REM HACERLO!
2620 GO SUB (EF*100)+3400
2630 RETURN
2700 REM **** PEDIR/PLANTARSE/DOBLAR ****
2710 GO SUB 800: IF EF=2 OR EF=3 THEN RETURN: REM
     COMPROBAR PONTOON/ROYAL PONTOON
2720 GO SUB 700: PRINT " PEDIR/PLANTARSE/DOBLAR ";
2725 LET A$=INKEY$: IF A$="" THEN GO TO 2725
2727 IF A$<>CHR$ 13 THEN PRINT A$
2730 IF A$="S" THEN GO SUB 2800: IF CS=0 THEN
     RETURN
2733 IF A$="S" AND CS=1 THEN GO TO 2700:REM NO SE
     PUEDE PLANTAR
2750 IF A$<>"T" THEN GO TO 2700: REM ERROR DE
     ENTRADA
2760 LET FL=0:LET PL=1:GO SUB 1300: REM
     REPARTIR
2770 GO SUB 800: IF EF=1 THEN GO TO 2700: REM OTRA
     VEZ?
2780 RETURN
2800 REM **** PLANTARSE ****
2810 LET CS=1: GO SUB 800: REM EVALUAR
2820 IF (T(PL,2)>=17 AND T(PL,2)<22) OR T(PL,1)>=17
     THEN LET CS=0: RETURN
2825 GO SUB 700: PRINT FLASH 1;"NO PUEDES PLANTARTE
     CON MENOS DE 17!"
2830 FOR L=1 TO 300: NEXT L
2840 RETURN
3500 REM MENOS DE 21
3505 LET PV=1: LET PS=T(1,2): IF PS>21THEN LET
     PS=T(1,1)
3510 GO SUB 700: PRINT "MENOS DE 21":
     RETURN
3600 REM **** ROYAL PONTOON ****
3605 LET PV=4
3610 GO SUB 700: PRINT "ROYAL PONTOON":
     RETURN
3700 REM **** PONTOON ****
3705 LET PV=2
3710 GO SUB 700: PRINT "PONTOON": RETURN
3800 REM **** BUST ****
3805 LET PV=0
3810 GO SUB 700: PRINT "BUST": RETURN
3900 REM **** JUEGO DE CINCO NAIPES ****
3905 LET PV=3
3910 GO SUB 700: PRINT "JUEGO DE CINCO NAIPES":
     RETURN
```

Commodore 64

```
120 GOSUB 2600:REM PEDIR ETC
2600 REM **** PEDIR APOSTADOR ETC EVALUAR ****
2610 GOSUB 2700:REM HACERLO!
2620 ON EF GOSUB 3500,3600,3700,3800,
     3900
2630 RETURN
2700 REM **** PEDIR/PLANTARSE/DOBLAR ****
2710 GOSUB 800:IF EF=2 OR EF=3 THEN RETURN: REM
     COMPROBAR PONTOON/ ROYAL PONTOON
2720 GOSUB 700:PRINT" PEDIR/PLANTEARSE/
     DOBLAR ";
2725 GET RESP$:IF RESP$="" THEN 2725
2727 IF RESP$<>CHR$(13) THEN PRINT RESP$
2730 IF RESP$="S" THEN GOSUB 2800:IF CS=0 THEN
     RETURN
2733 IF RESP$×"S" AND CS=1 THEN 2700:REM NO SE
     PUEDE PLANTAR
2750 IF RESP$<>"T"THEN 2700:REM ERROR DE
     ENTRADA
2760 FL=0:PL=1:GOSUB 1300:REM REPARTIR
2770 GOSUB 800:IF EF=1 THEN 2700:REM OTRA
     VEZ?
2780 RETURN
2800 REM **** PLANTARSE ****
2810 CS=1:GOSUB 800:REM EVALUAR
2820 IF (TT(PL,2)>=17 AND TT(PL,2)<22) OR
     TT(PL,1)>=17 THEN CS=0:RETURN
2825 GOSUB 700:PRINTCHR$(28);"NO PUEDES PLANTARTE
     CON MENOS DE 17"
2830 FOR DL=1 TO 1000:NEXT DL
2840 RETURN
3500 REM **** MENOS DE 21 ****
3505 PV=1:PS=TT(1,2):IF PS>21 THEN PS=
     TT(1,1)
3510 GOSUB 700:PRINT"MENOS DE 21":RETURN
3600 REM **** ROYAL PONTOON ****
3605 PV=4
3610 GOSUB 700:PRINT"ROYAL PONTOON":
     RETURN
3700 REM **** PONTOON ****
3705 PV=2
3710 GOSUB 700:PRINT"PONTOON":RETURN
3800 REM **** BUST ****
3805 PV=0
3810 GOSUB 700:PRINT"BUST":RETURN
3900 REM **** JUEGO DE CINCO NAIPES ****
3905 PV=3
3910 GOSUB 700:PRINT"JUEGO DE CINCO NAIPES":
     RETURN
```

Caroline Clayton

Preguntas al directorio

Desde buscar en un directorio hasta imprimir un archivo, las instrucciones residentes del MS-DOS se ocupan de casi todas las eventualidades

Los sistemas MS-DOS normalmente se cargan tan sólo con el encendido (desde un disco rígido) o insertando un disco de sistema flexible. Muchas máquinas poseen teclados tipo IBM y en éstas se puede efectuar una reinicialización o arranque en caliente en cualquier otro momento mediante el método "Control-Alt-Delete" tradicional del IBM PC (se mantienen pulsadas dos teclas del lado izquierdo del teclado, etiquetadas CTRL y ALT, mientras simultáneamente se pulsa la tecla DEL [*delete*] del lado derecho). La última operación, después de que se ha cargado el MS-DOS y llevado a cabo la inicialización del sistema, es para consultar la unidad por defecto en busca de un archivo *batch* denominado AUTOEXEC.BAT. Si está presente, las instrucciones que contiene se ejecutarán automáticamente como si se las hubiera entrado desde el teclado.

Podríamos, por ejemplo, escribir un archivo AUTOEXEC.BAT simple que contuviera la instrucción:

```
MENU
```

donde MENU.COM (o MENU.EXE) fuera un programa de aplicación adecuado (un archivo EXE [de *executable*: ejecutable] o COM [de *command*: instrucción]). Este programa se ejecutaría automáticamente cada vez que se cargara el sistema con este disco en particular; esto se conoce como sistema *turnkey* (giro de llave).

El MS-DOS (y el PC-DOS) posee más potencia y mayor flexibilidad, y todas las utilidades "transitorias" se proporcionan como archivos EXE o COM separados. En este capítulo nos concentraremos, sin embargo, en el núcleo de instrucciones "residentes" comunes a todas las versiones de MS-DOS y PC-DOS. Éstas están disponibles a través del intérprete de línea de comando DOS, o "caparazón", denominado COMMAND.COM (la única parte del DOS que queda "visible" para el usuario).

De todas las utilidades del sistema, quizá la que se utiliza con mayor frecuencia sea la instrucción de directorio. Ésta lista todos los archivos de un disco o, a partir del DOS 2 en adelante, de un *directorio* (una porción con nombre y restringida de todo el espacio del disco). La forma básica de la instrucción no es más que:

```
dir
```

Al igual que en CP/M, sobre el cual se modeló íntimamente el DOS 1, esta forma simple lista todos los archivos del disco conectado actualmente (por "defecto"). Si se requiriera un listado del directorio de otra unidad, esto se especificaría como un parámetro de línea de comando. Nuevamente, esto es idéntico al uso del CP/M y también sigue las mismas especificaciones para referencias ambiguas a archivos: ? representa cualquier carácter individual, y se puede utilizar * ya sea para un grupo de caracteres o bien, posiblemente, para ningún carácter en absoluto.

En el nombre de archivo primario se admiten entre uno y ocho caracteres, y hasta tres caracteres (o ninguno) para la extensión o nombre secundario; nuevamente, tal como en CP/M. No obstante, cuando el listado aparece en la pantalla se observa una diferencia radical tanto en el formato utilizado como en la cantidad de información visualizada. El MS-DOS lista no sólo los nombres de los archivos, sino también el tamaño de éstos (en bytes) y la fecha y hora en que se escribió cada archivo en el disco.

He aquí un listado de directorio típico:

```
A>dir

Volume in drive A has no label
Directory of A:

COMMAND   COM  22672   9-03-85  11:36a
AUTOEXEC  BAT    128  22-07-85   5:15p
WP        EXE  73156  12-10-84  12:07p
WPMSG     OVR  38269  27-09-84  12:02p
WPINST    EXE  56385   3-10-84  12:04p
WCOUNT    PAS   4986   1-08-85   3:11p
WCOUNT    OBJ   3874   1-08-85   3:12p
WCOUNT    EXE   8385   1-08-85   3:14p
WCOUNT    BAK   4732  31-07-85  11:23p
MARY1     TXT    634   7-08-85  11:05a
INVOICE1  885    477  21-08-85   9:36a
MARY2     TXT    938  15-08-85  12:47p
```


Hil Rudge, cortesía de "Microscope"

En muchos sistemas quizá la fecha aparezca en formato norteamericano (mes-día-año), pero las versiones de DOS recientes se ciñen a la convención europea, además de admitir juegos de caracteres especiales para las lenguas europeas y escandinavas. Si bien los tamaños se dan en bytes, los archivos suelen ocupar más espacio en el disco. Cada archivo utiliza una cierta cantidad de sectores completos, generalmente 512 bytes cada uno.

El DOS también visualiza el número de archivos y el espacio restante en el disco. Esto combina muchas de las características de la utilidad CP/M externa ("transitoria") STAT.COM con las de la instrucción dir residente y, junto con el sello de hora y fecha de la creación del archivo, se convierte en una instrucción mucho más útil. El motivo fundamental de esta funcionalidad adicional es la supresión de la barrera de 65 Kbytes de memoria.

Los sistemas CP/M estaban casi invariablemente restringidos a este espacio máximo de direccionamiento, a menos que implementaran sofisticados sistemas de paginación de memoria. Cada puñado de bytes que empleaba el OS le "robaba" al usuario igual cantidad de RAM libre, de modo que en el código del sistema sólo se incorporaba lo esencial como instrucciones residentes. Se obtenía información detallada (como el tamaño del archivo y atributos) mediante la ejecución de un programa separado (como STAT) a modo de utilidad transitoria.

El MS-DOS también posee una gama amplia y versátil de estos programas, pero merced al tan acrecentado espacio de memoria de los sistemas de 16 bits (lo normal es 256 Kbytes o más), el sistema residente puede ser mucho más grande y, por tanto, contener muchas facilidades más potentes. La versión 3 del DOS fácilmente puede requerir más de 60 Kbytes de RAM; la cantidad exacta depende en parte del distribuidor que suministre el sistema. El MS-DOS da por sentada la existencia de un reloj del sistema e, incluso aunque no haya un verdadero dispositivo de hardware (con apoyo de batería) para seguir funcionando cuando se apague el ordenador, habrá disponible una simulación por software, por lo general operando con un método de interrupción continua.

En este caso, el reloj de software se inicializará

Instrucciones residentes del MS-DOS

Instrucción	Función	Interrs.	Uso	
copy	copiar	/a/b/v	copy	{unidad:} nombre-con máscara {destino}
date	visualizar /fecha est.		date date	{dd-mm-aa} {mm/dd/aa} (versión USA)
del	borrar arch.		del	{unidad:} {n.-con-máscara}
dir	listar arch.	/p/w	dir	{unidad:} {n.-con-máscara}
erase	ver del			
ren	cambiar nombre archivos		ren	{unidad:} nombre1 nombre 2
rename	ver ren			
time	visualizar /hora est.		time	{hh:mm:ss}
type	listar archivos texto		type	{unidad:} nombre-archivo

Clave:

{ } los elementos que aparecen entre llaves son opcionales

| una barra vertical separa alternativas

carácter ::= cualquier carácter ASCII excepto",.?*<>;:[]/\|
nombreprimario::=carácter{carácter} (máximo ocho)
nombresecundario::={carácter} (máximo tres)
nombrearchivo::=nombreprimario{.nombresecundario}
nombre-con-máscara::=nombrearchivo conteniendo caracteres "de máscara" (*y?)
dispositivo::=CON | LST | PRN | AUX | LPT1 etc.
destino::=nombrearchivo | dispositivo
unidad::=A | B | etc. (generalmente hasta P)

Caroline Clayton

en, por ejemplo, 1-01-85 12:00:00 y se debe ajustar manualmente cuando se carga el sistema a través de las instrucciones DOS date y time. Éstas visualizan la hora (o fecha) actual e invitan a entrar nuevos valores desde el teclado, que deben responder a los mismos formatos consignados con anterioridad, aunque no necesariamente completos. La interacción también se puede abreviar entrando nuevos valores en la línea de instrucción. De modo que, por ejemplo:

```
time 14
```

establecería al reloj del sistema en 14:00:00 (los dos últimos dígitos representan centésimas de segundo, si bien a menudo la resolución de tiempo no será tan alta como esto). Todo archivo que se cree poco después se estampará con los valores actuales del sistema y podría aparecer en el listado del directorio de esta forma:

```
NUEVOARCHIVO   TXT   512   21-08-85   2:05p
```

A todos los programas MS-DOS se les da la extensión EXE (*executable*: ejecutable) o COM (*command*: instrucción), siendo la única diferencia las restricciones en cuanto a las posiciones de memoria en las cuales se pueden volver a localizar y ejecutar. En términos generales, los archivos EXE son más flexibles y se pueden cargar en los límites de un párrafo

Un hombre de suerte

La historia de cómo Bill Gates, de Microsoft, se aseguró el contrato para suministrar a IBM un OS de 16 bits (MS-DOS) es un tanto inusual. Habiendo sido contactada por IBM, inicialmente Microsoft no estaba segura de si podría aceptar el contrato para producir el OS. Por consiguiente, IBM decidió hacer una visita a Digital Research para hablar del asunto con Gary Kildall, quien era conocido por haber escrito el CP/M. El día de la visita, sin embargo, Kildall no estaba en su despacho. Tras esperarlo durante dos horas, la delegación de IBM se marchó y posteriormente se le concedió el contrato a Microsoft. De este modo, un contrato que conduciría a la creación de un estándar industrial se le concedió a una empresa a la cual sólo se conocía por una versión de BASIC y que contaba con poca experiencia en tecnología de sistemas operativos

(palabra del ordenador), mientras que los archivos COM deben residir dentro de los límites de un *segmento* (al comienzo de una zona o segmento de 64 Kbytes, como las convenciones de la familia Intel 8086). Se utilizan otros nombres de archivo secundarios con significados especiales, en ocasiones por razones de mera conveniencia, de modo que con frecuencia a los documentos, cartas y otros archivos de texto se les suele dar la extensión .TXT (o .DOC) a modo de ayuda mnemotécnica. Los archivos objeto en código máquina con formato Intel estándar (reubicable), tendrán la extensión .OBJ, designándose al código fuente con las extensiones .PAS, .FOR o .C usuales para programas en PASCAL, FORTRAN y C.

Estudiando un poco más el listado del directorio, podemos deducir que el programa WCOUNT.EXE se originó como un archivo fuente en PASCAL (WCOUNT.PAS se guardó a las quince horas y 11 minutos de la tarde del 1.º de agosto). Éste se compiló como un archivo .OBJ de formato Intel estándar alrededor de un minuto después, y dos minutos más tarde se montó para producir el programa ejecutable (EXE) final. Todavía se detectan indicios de la versión previa (WCOUNT.BAK) archivada como copia de seguridad (BAK) por el procesador de textos (WP.EXE), que obviamente causó bastantes problemas, a juzgar por el hecho de que el programador parece haber estado trabajando hasta altas horas de la noche la semana anterior (el 31 de julio). Quizá, a juzgar por su nombre, podríamos aventurar que este programa cuenta las palabras de un archivo de textos; ¡y toda esta información la hemos obtenido con sólo un rápido estudio de la información que ofrece dir!

La instrucción tiene un par de ases bajo la manga que la diferencian más aún de su equivalente CP/M. Éstos asumen la forma de *interruptores*, u opciones añadidas en la línea de instrucción. El interruptor P da una página (o pantalla llena) de archivos por vez, haciendo una pausa hasta que se pulsa una tecla antes de proseguir con el listado. Esto es especialmente útil en los modernos discos de gran capacidad con los que se podrían almacenar centenares de archivos en un dispositivo o directorio. El otro interruptor, W, permite la supresión de la información detallada, y hace que sólo los nombres de archivo se listen en formato W*ide* (ancho), de cinco por línea. Muchas instrucciones DOS, internas y externas, poseen tales interruptores y su condición se indica precediendo a *cada uno* con una barra inclinada:

```
dir c:*exe/p/w
```

Esto podría abarcar hasta 115 (5×23) archivos .EXE por "página" de pantalla.

Otra instrucción que se comporta como su equivalente de CP/M es la que se utiliza con mayor frecuencia para visualizar el contenido de un archivo de textos en la pantalla (o impresora). La sintaxis es la siguiente:

```
type {dispositivo:} nombrearchivo
```

De modo que, por ejemplo, para listar un programa en MODULA-2 llamado RATON.MOD en el dispositivo de disco B, entraríamos:

```
type b:raton.mod
```

Al igual que en los sistemas CP/M, generar un Control-P antes del retorno de carro activará la impresora, en caso de que deseemos un listado por impresora.

Existe un método alternativo al del CP/M, que en cierto modo es preferible y, por cierto, más flexible. Mientras que el CP/M utiliza PIP.COM como programa de sistema transitorio para transferencia de datos, el MS-DOS tiene una utilidad COPY muy potente incorporada como instrucción residente. En su forma más simple, se la invoca mediante:

```
copy {dispositivo:} nombrefuente {destino}
```

donde el destino puede ser otro nombre de archivo o un dispositivo del sistema como CON (la consola), LIST o PRN (el dispositivo de listado estándar o impresora), respectivamente. De modo que, para obtener una salida impresa:

```
copy b:raton.mod prn
```

conseguiría lo mismo que la instrucción type de arriba, pero sin tener que utilizar Control-P para evitar que en el listado aparezca ninguna otra cosa a excepción del contenido del archivo. Con la instrucción type se añadiría un aviso del sistema (como, por ejemplo, A>).

La instrucción copy puede tomar tres interruptores opcionales: /a copia archivos ASCII utilizando una marca de final del archivo (Control-Z), /b emplea el final físico del archivo, ignorando cualquier Control-Z, y /v verifica cada transferencia de datos (conmutable permanentemente con verify activado). Esto es posible porque el MS-DOS lleva un registro del tiempo que permanece cada archivo en el directorio. Si no se especifica el destino, copy da por sentado que usted se refiere a un archivo del mismo nombre que la(s) fuente(s), pero en la unidad por defecto (como copy c:*.txt/a/v).

Esta potente instrucción residente tiene muchas otras posibilidades. Entre las más interesantes está la capacidad de concatenar archivos. Si decimos:

```
copy b:*.txt grandoc.txt
```

todos los archivos de texto del dispositivo B se concatenarán en el archivo grandoc.txt en la unidad por defecto (por el orden del directorio). Si deseamos controlar el orden o especificar archivos con extensiones diferentes, se puede usar el signo +:

```
copy c:x1.wrk+x2.dat+b:x3.frm xxx.new
```

Omitiendo el destino se obtiene el añadido a un archivo existente:

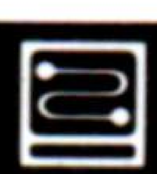
```
copy page1.txt+page2.txt+page3.txt
```

con el contenido de page2.txt y luego de page3.txt añadiéndose al final de page1.txt, conservando este nombre para el archivo resultante.

Hay un editor del sistema (denominado EDLIN.EXE), del cual nos ocuparemos en el próximo capítulo, junto con algunas de las otras «instrucciones» transitorias. Mientras tanto, se puede crear un archivo de texto sin la ayuda de ninguna clase de procesador de textos con tan sólo impartir la instrucción:

```
copy con nuevoarchivo.txt
```

Tras entrar el texto deseado, se cierra el archivo y se lo escribe en disco entrando Control-Z (la marca de final de archivo). A diferencia de pip (CP/M), se lo debe enviar seguido de un ENTER<CR>).



Al pie de la letra

Prosiguiendo nuestro estudio de las instrucciones del 68000 de Motorola, consideraremos la instrucción de comparación

Estrechamente relacionada con la instrucción SUB tenemos la instrucción CMP (comparación), cuya importancia estriba en que, sea cual sea el área de programa en que estemos trabajando, siempre habrá elementos de decisión en el diseño de ese programa. Consideremos el siguiente diseño en alto nivel, que incluye un elemento *pseudo-code* de decisión:

```
If inputchar='N' then
   cleararray
end
```

Aquí la decisión If se codificará así en la fase de implementación del proyecto:

```
CMP.B #'N',D0      Compara el byte de entrada
                   con N
BNE    NOTSAME     Va a NOTSAME si es
                   diferente
```

Cuando se ejecuta la instrucción CMP, la fuente es restada del destino y se cambian sólo los códigos de condición, sin que sea afectado el destino (como ocurre, por ejemplo, con la instrucción SUB). La bifurcación condicional, BNE, que sigue provocará una bifurcación a la etiqueta NOTSAME si el resultado de la recta de D0-'N' no es cero.

Veamos otros dos ejemplos que emplean atributos diferentes de datos:

```
CMP.W SPEED,D3     Compara contenido palabra de
                   posición SPEED con D3
CMP.L D1,D2        Compara pals. largas completas
                   contenidas en D1 y D2
```

Se notará que los campos de destino en estos ejemplos son registros de datos, y que de hecho puede utilizarse cualquier modo de direccionamiento como modo de direccionamiento fuente. Si es necesario direccionar cualquier otra forma de destino, se usan diferentes instrucciones CMP. Por ejemplo:

```
CMPA   BETTY,A3    Compara el contenido de la
                   posición BETTY con A3
```

En realidad es posible el uso de cualquier modo de direccionamiento fuente. Sin embargo, la forma inmediata que sigue sólo permite modos alterables de datos.

```
CMPI    #3,(A4)    Compara cualquier A4 que se
                   apunte con 3
```

Una última variante de CMP digna de mención es la instrucción CMPM. Por ejemplo, CMPM (A2)+,(A3)2 comparará el contenido de las posiciones de memoria apuntadas por A2 y A3 y después posincrementará los punteros. Por ello, esta instrucción puede ser utilizada para comparar, por ejemplo, palabras clave almacenadas con caracteres que están en un buffer de entrada, y todo eso también en una palabra. He aquí un ejemplo:

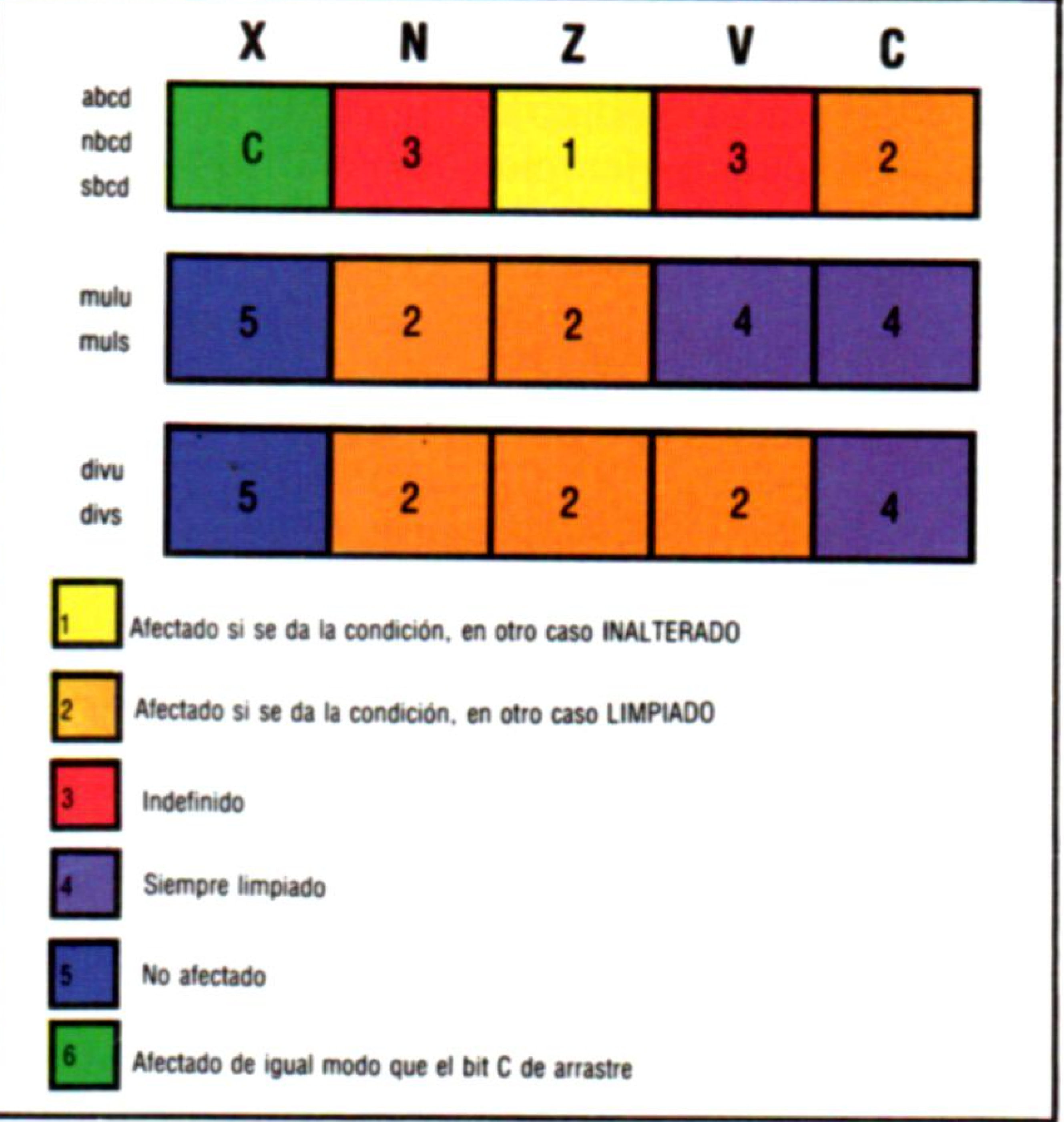
```
LEA      KEYS,A2     Establece el primer puntero
                     hacia las palabras claves
LEA      BUFFER,A3   Y el puntero del buffer
CHECK    CMPM.B      Compara las dos posiciones de
         (A2)+,      memoria
         (A3)+
BEQ      CHECK       Va comparando hasta que
                     sean diferentes
```

El programa en total ocupa sólo siete palabras.

Otro aspecto ulterior es que la instrucción CMPM sólo se permite con el modo postincremento de direccionamiento; así, si se desea cualquier otra forma, entonces un operando ha de ser cargado en el registro de datos en primer lugar y emplear la instrucción CMP.

Dos sencillas instrucciones aritméticas son NEG y EXT. La instrucción NEG (negación) resta de 0 el registro de datos del operando, es decir, forma el valor negativo en complemento a dos del contenido del registro de datos (no se permite ningún otro modo de direccionamiento). Así, por ejemplo, si D0=1111 1010 (−6, en decimal) y ejecutamos NEG.B D0, ocurrirá que D0 contendrá 0000 0110 (6, en decimal). Empleos típicos de esta instrucción incluirán la obtención del valor absoluto de un operando de datos comprobando primeramente el valor negativo y después dándole esa forma negativa. Existe también una forma ampliada de la instrucción, que incluye el bit X, llamada NEGX.

La instrucción EXT se emplea para ampliar con el bit signo del operando de datos el tamaño del operando más grande. Así, si se ejecuta EXT.W D0



Cambio de estado
Los efectos de las instrucciones DIV, MUL y BCD sobre el registro de estado son los que aquí se ilustran. Es importante observar que en algunos casos la ausencia de una condición no se traduce en la limpieza del flag correspondiente, que quedará inalterado y, si se emplea de testio, puede dar una falsa lectura (la que corresponde a una operación previa)

donde D0=1111 0101 (FA, en hexa) dará D0=FFFA (en hexa). Esta instrucción es útil cuando se trabaja con números de precisión múltiple, en especial cuando se incluyen operandos más grandes de datos, como pueden ser las instrucciones de multiplicación y división.

Interpretación de patrones de bits

Antes de seguir con el examen de instrucciones más complicadas, debemos considerar el empleo de los modelos binarios de bits en un operando de datos de un byte. Por ejemplo, cuando D0=1001 0110 podemos suponer que se trata de un entero con un valor de –106 (recuerde que el valor se obtiene haciendo una inversión y sumando la unidad). Sin embargo, si el número no tiene signo y se asume el intervalo global binario con enteros positivos, el valor sería 96 (en hexa), o sea 150 en decimal (9×16+6×1). Los números sin signo son útiles si sabemos que sólo vamos a necesitar un intervalo grande positivo. Por ejemplo, el intervalo de direcciones de un ordenador puede considerarse como un intervalo de números positivos sin signo, y a condición de que no operemos con esos números mediante operadores con complemento a dos, así pueden tomarse.

Pero hay todavía una tercera interpretación del modelo de bits dado; es el sistema llamado BCD (*binary coded decimal*: decimal codificado en binario). Se trata de una codificación muy adecuada en la que todo grupo de cuatro dígitos binarios es considerado como el código de un dígito decimal. En este caso, nuestro ejemplo (D0=1001 0110) tendría en BCD el valor de 96 (1001=9 y 0110=6).

Debemos hacer tres importantes observaciones sobre esta codificación. Primera, lo fácil que es convertir números incluso muy grandes de decimal a BCD, o viceversa. Por ejemplo, el decimal 9 631 en BCD se escribe 1001 0110 0011 0001. La conversión, como se ve, es muy sencilla.

La segunda observación es que también resulta fácil definir la precisión de los números codificados de esta manera. La tercera se refiere a que codificamos dígitos del 0 al 9, por lo que los códigos sobrantes no tienen validez (es decir, los códigos que van desde el 1010, que es 10 en decimal, al 1111, que es 15 en decimal).

La importancia de estas observaciones se hará evidente cuando estudiemos las distintas instrucciones aritméticas del 68000. Veámosla por un momento en la operación de multiplicar. Si multiplicamos dos operandos de 16 bits, el modelo de bits resultante puede tener 32 bits de longitud. Usted mismo puede comprobarlo, pero entienda que para una palabra de n bits de longitud, el producto del número más grande, $2^{(n-1)}$, será $2^{2(n-1)}$, o sea, dos veces la longitud de la palabra original.

Para la instrucción de multiplicar en binario, disponemos por ello de dos operandos de 16 bits que dan como resultado 32 bits. Puesto que podemos operar con números con y sin signo, tenemos dos instrucciones independientes, la MULU (*multiply unsigned*: multiplicación sin signo) y la MULS (*multiply signed*: multiplicación con signo). Ambas instrucciones multiplican los dos operandos y ponen un resultado de 32 bits en el registro de datos de destino. De nuevo se observará que sólo se permiten modos de direccionamiento de datos para el operando fuente.

Así, por ejemplo, MULU #20,D0 cuando D0=XXXX 0003 (la X significa aquí 0 o 1) dará D0=0000 003C. Se notará que se emplea el registro entero de datos de 32 bits aun cuando no sea necesario en este ejemplo particular.

Del mismo modo, cuando D0=XXXX FFFF, MULU #$10,D0 dará como resultado D0=000F FFF0 pero con MULS se obtendría D0=FFFF FFF0. Esto es así porque el resultado es de signo ampliado a todos los 32 bits. Se observará que es posible comprobar fácilmente estos resultados dado que el multiplicador de 10 hexa equivale a un desplazamiento a la izquierda de 4, o a una multiplicación por 16.

Una consideración final sobre la instrucción para multiplicar se refiere a los códigos de condición. Los flags N y Z se activan según sea el resultado, pero los bits V y C son puestos a 0. Aunque no es posible que con operandos de palabras y resultados de una palabra de longitud obtengamos un desbordamiento, es interesante conocer si el resultado de palabra se desbordaría, porque entonces se sabrá fácilmente si el resultado puede ser truncado a una palabra en caso de necesidad.

Veamos un fragmento típico de programa que emplea la instrucción MULS. Supongamos que se desea convertir caracteres ASCII representativos de números decimales en palabras binarias. La primera operación será convertir el carácter de entrada en binario y sumar seguidamente este patrón de bits en el acumulador binario. Antes de efectuar la suma tenemos que asegurarnos de que las entradas anteriores fueron multiplicadas por 10 (pues se trata de entradas decimales). Así, por ejemplo, un posible programa sería éste:

```
SUB.B  #'0',D0   Forma el binario del carácter
                 decimal
MULS   #10,D5    Multiplica la suma previa por 10
                 decimal
ADD.W  D0,D5     Suma el nuevo valor
```

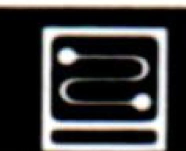
En este ejemplo el carácter de entrada está en D0 y la nueva palabra binaria que representa en binario los caracteres de entrada está en D5.

Examinemos ahora la instrucción de dividir, DIVU (para números sin signo) y DIVS (para números con signo). Ambas instrucciones toman el entero de 32 bits que está en el registro de datos de destino y lo dividen por el operando fuente de 16 bits. El resultado se pone en el registro de datos de destino y en los 16 bits inferiores colocando cualquier resto en los 16 bits más significativos. Por ejemplo, si D0=0000 0005 (en hexa):

```
DIVU   #3,D0
```

dará D0=0002 0001 (1 con resto 2). Observe que el operando destino es un operando completo de 32 bits, lo que significa que primero puede necesitarse una palabra larga simple, o una larga EXT.L ampliada con signo. Dado que es posible el desbordamiento (p. ej., dividiendo por la unidad una palabra entera de 32 bits obtendremos como resultado algo más de 16 bits), se tendrá en cuenta oportunamente el bit de desbordamiento.

Un sencillo algoritmo que nos permita contar todos los números de una lista de palabras terminados en cero podría ser el siguiente:



```
                ORG $1000
          LEA   LIST1,A0
PUNTERO                     establece
                              dirección
          CLR.L D0
          CLR.L D1          borrado total
LOOP      ADDQ  #1,D1       contador bucle
                            cuenta núm.
          ADD.W (A0)+,D0      palabras
          TST   (A0)        y suma
                            comprueba
          BNE   LOOP          fin de lista
          DIVU  D1,D0
                            divide suma
          TRAP  0             por contador
LIST1     DC.W  1,2,3,4,0
```

Se notará que se han empleado borrados de largas palabras en la parte de inicialización de este programa, ya que la suma será tratada como operando completo de 32 bits. En D1 se coloca un contador y la palabra apuntada A0 se añade a D0 con el direccionamiento de postincremento. Seguidamente se comprueba por medio de la instrucción TST la siguiente palabra que está en LIST1. Esta instrucción se limita a colocar códigos de condición listos para la instrucción BNE, y no altera operando alguno.

La instrucción TST es necesaria porque la instrucción anterior ADD afectará los códigos de condición según sea el resultado de sumar los operandos de datos en D0, y no según el valor de los datos cargados por el puntero A0. La instrucción BNE (B*ranch if* N*ot* E*qual to zero:* bifurcar si no es igual a cero) provocará la vuelta a LOOP mientras no sea cero el siguiente elemento de la lista. Cuando ocurra que es igual a cero, D0 contendrá la suma y D1 el número de elementos no cero (el contador de datos).

Cuando se haya ejecutado el bucle (LOOP) cuatro veces, los valores contenidos en los registros de datos serán:

```
D0=0000 000A (en decimal,10)
D1=0000 0004
```

Por tanto, una vez ejecutada la instrucción DIVU, entonces D0 contendrá 0002 0002 (10 ÷4=2 y me llevo 2).

Una última observación sobre las instrucciones para dividir es que una división por cero provocará un *trap* (una interrupción software en el monitor del sistema), ya que un número infinito ciertamente no es representable en 16 bits. Si no queremos caer en esta "trampa", hemos de establecer un testigo cuando el divisor sea cero.

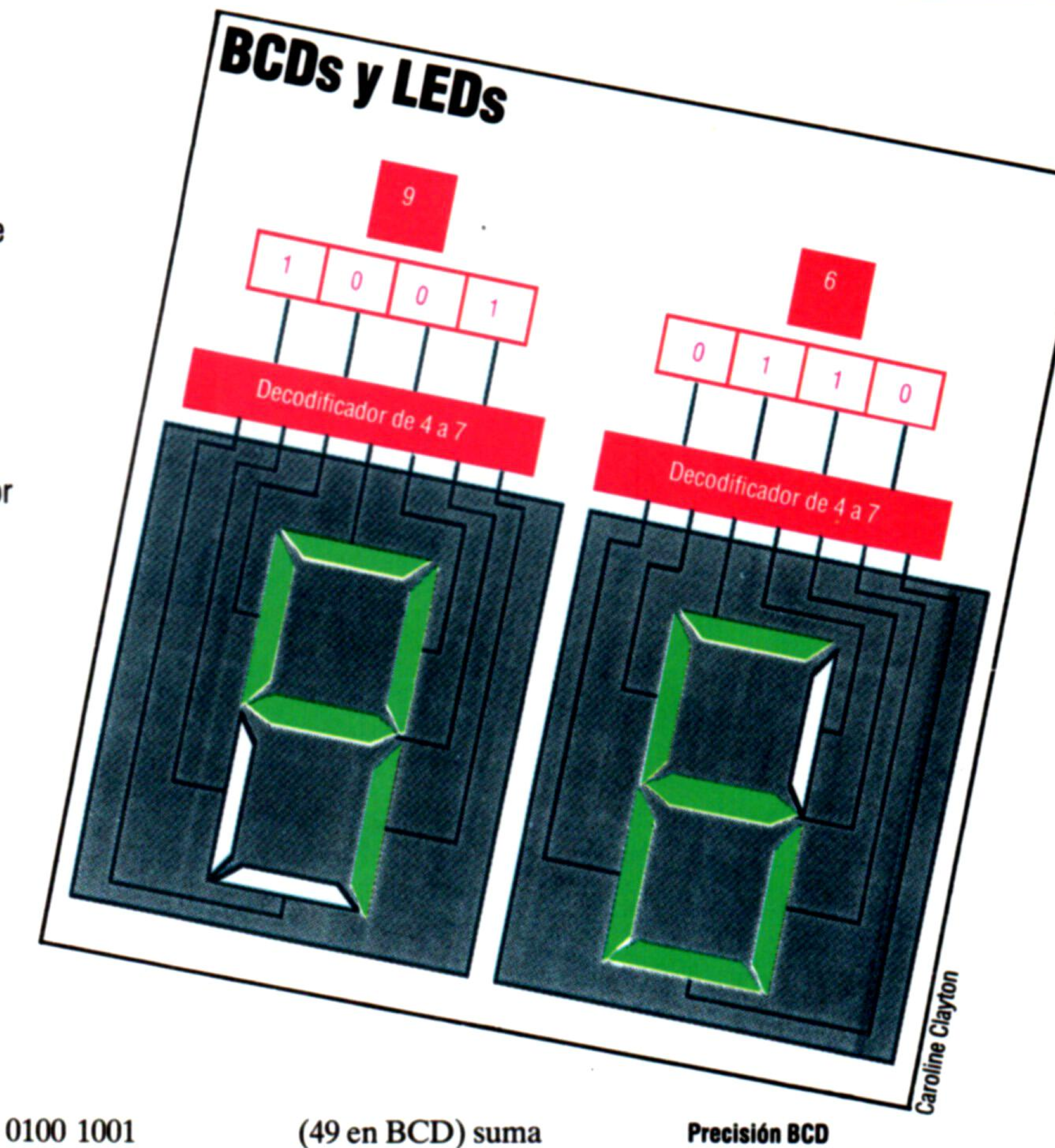
Por ejemplo:

```
TST     D1
BEQ     ERROR
DIVU    D1,D0
```

Ya hemos visto la conveniencia del BCD para la representación de números decimales. Pero se ha de notar que un dígito BCD válido corresponde a un dígito en hexa (4 = 0100 binario = 4 hexa = 4 BCD), por lo que las constantes BCD equivalen a constantes hexa. Por ejemplo:

```
MOVE.B     #$54,D0   pone 54 BCD en D0
```

Pero la analogía acaba aquí. Al sumar dos dígitos BCD en aritmética binaria, por ejemplo, obtendremos una respuesta incorrecta:



Precisión BCD
Lo mismo que para operar con enteros con y sin signo, el 68000 posee un juego de instrucciones para la manipulación de datos tomados en forma BCD (decimal codificado en binario). Esto es especialmente útil cuando se necesita una precisión aritmética total. Pero también ofrece otras ventajas en distintas aplicaciones. Por ejemplo, en el manejo de visualizaciones LED de siete segmentos. El hardware que decodifica los datos en BCD emplea un decodificador de líneas de 4 por 7, como en el dibujo, resultando más sencillo que la decodificación de números binarios

```
0100 1001      (49 en BCD) suma
0000 0001      binaria
               (1 en BCD)
---------
0100 1010
```

Obtenemos un dígito BCD ilegal como dígito BCD menos significativo. Esto no debe preocuparnos, sin embargo, dado que el 68000 posee un grupo de instrucciones BCD para sumar (ABCD), restar (SBCD) y negar (NBCD).

Nuestro estudio de las instrucciones BCD se limitará a la instrucción ABCD, que suma el byte fuente (dos dígitos BCD) al byte destino, con el bit X, poniendo la suma en el destino. Los únicos modos de direccionamiento permitidos son los pares de registros de datos y los pares de registros de direcciones predecrementadas. Así, por ejemplo, si D0=44 BCD y D1=01 BCD, entonces tras ejecutar

```
ABCD     D0,D1
```

D1 contendrá 45 BCD. Si el bit X que está en el SR estaba activado (equivalente al arrastre para BCD), el resultado sería 46. Igualmente, si sumamos 1 a 99 en D0, el resultado sería 00 con el bit activado en el registro de estado. Con esto se puede colegir que si bien estamos limitados a operandos de un byte, podemos ampliar fácilmente la precisión de nuestros cálculos empleando el bit X para llevar el arrastre a componentes del byte más significativo.

Los códigos de condición X, C y Z se activan para todas las instrucciones BCD. Sin embargo, es de notar que la instrucción NBCD (Negar en BCD) permite modos de destino alterables de datos preferentemente a pares predecrementados o simples datos.

Pantallas mil



Tesoro enterrado
Quo vadis, de Softek, es una aventura en gráficos con un premio real. Trasladándose por los escenarios de la aventura y resolviendo tres acertijos, el jugador puede recoger pistas que lo orientarán en la búsqueda de un cetro de oro y plata que se halla enterrado

"Quo vadis" introduce al jugador en un misterioso y fascinante ambiente situado en un mítico medievo, ¡y posee 1 024 pantallas!

El concepto "aventura recreativa", que combina elementos de los programas de aventuras y recreativos tradicionales, ha dado un nuevo giro desde que algunas empresas de software han centrado su interés en otra tradición que ha alcanzado creciente difusión desde la publicación del libro *Masquerade* (Mascarada), de Kit Williams. En esta obra se proporcionaban a los lectores numerosas pistas visuales y escritas que conducían a la ubicación real de una liebre de oro, enterrada en algún lugar de la campiña inglesa. La primera empresa que introdujo esta idea en el mercado de software para ordenadores fue Automata con su juego *Pimania*, que ofrecía como premio un objeto de oro. Después vino *Hare raiser* (Criador de liebres), un juego por ordenador similar en concepto al *Masquerade* y que incluso ofrecía un premio idéntico, que se le compró a su descubridor original y luego se volvió a ocultar.

Softek continuó esta tradición de ofrecer premios en su juego de aventuras *Quo vadis*, que ofrecía un cetro de oro y plata para la primera persona que resolviera los enigmas y escapara del juego. La esfera de acción del programa constituía una sensación adicional, dado que ofrecía una superficie de juego que cubría 1 024 pantallas diferentes.

El juego es una aventura recreativa en la que el usuario asume el papel de un caballero que busca un cetro que se halla escondido en algún punto de un complejo sistema de cavernas subterráneas. Es este laberinto subterráneo lo que ocupa las 1 024 pantallas; está constituido, además, por 118 cuevas, cada una de las cuales es más grande que la superficie total de juego de muchas de las aventuras recreativas normales.

Para desplazarse a través de las cavernas es preciso saltar de una saliente rocosa a otra o, a veces, ascender o descender trepando por unas cuerdas. Además hay que salvar diversos peligros, como las simas de lava de la parte inferior de las cuevas, ya que caer en ellas puede acarrear fatales consecuencias. Con estas características de la plataforma del juego se combinan diversos elementos del juego de acción tradicional. Las cavernas están habitadas por 38 clases diferentes de monstruos, quienes atacan al caballero cuando éste penetra en una cueva, y se utiliza la palanca de mando para disparar en una de ocho direcciones.

Si el caballero resulta herido por los monstruos pierde puntos de energía, y cuando éstos hayan disminuido de 100 a 0 el juego termina. El caballero, no obstante, puede reaprovisionarse de energía mediante cofres de tesoros que encuentra en ciertos escenarios.

En algunas cavernas, los enigmas escritos sobre las paredes proporcionan pistas que pueden ser útiles para hallar el cetro.

Quo vadis es impresionante no sólo en virtud de su tamaño. El juego no se compone de pantallas separadas que se vayan superponiendo unas sobre otras, sino que la imagen se va desplazando con gran uniformidad para revelar un poco más del paisaje circundante. De este modo, da la impresión de penetrar lentamente en un mundo realista y misterioso.

Dada la inmensidad de *Quo vadis*, era inevitable que hubiera de resentirse algún aspecto del juego; no obstante, es justo afirmar que los gráficos y el sonido responden por completo a la media. Pero cuanto más se practica el juego los gráficos van proporcionando más atmósfera, y las brillantes antorchas, las velas mortecinas, las oscuras escaleras y las húmedas paredes contribuyen en gran medida a crear un convincente ambiente subterráneo.

El hecho de que un juego tan grande pueda retenerse en la memoria del Commodore 64 se debe atribuir a las potentísimas técnicas de compactación de código utilizadas en la programación. Sólo se emplean seis bytes para cada pantalla, dejando, por lo tanto, espacio suficiente para las rutinas del juego.

Poco después de la aparición del juego, Softek sacó en disco un *Quo vadis generator* para quienes considerasen insuficientes las 1 024 pantallas ya incluidas. La empresa afirma que mediante el uso de este disco ahora hay disponibles millones de pantallas diferentes.

Quo vadis: Para el Commodore 64
Editado por: Softek International, 12/13 Henrietta St., Covent Garden, London WC2E 8LH, Gran Bretaña
Formato: Cassette o disco
Palanca de mando: Necesaria


GRAN NOVEDAD EDITORIAL
PARA TI, PROFESIONAL EN ACTIVO.....
PARA TI, FUTURA SECRETARIA.....
enciclopedia de la
SECRETARIA
de
PLANETA - AGOSTINI
PIDELA EN TU QUIOSCO
o suscríbete ahora
llamando al (91) 415 97 12
... para estar al día
... para ser más eficaz
... para actualizar tus estudios
... para encontrar mejor empleo

10113
9 788485 822836